Where Myhill–Nerode Theorem Meets Parameterized Algorithmics

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
Contents

1 Decomposing the Input 3
   And running dynamic algorithms: a try to give a useful unifying view . . .

2 The Concept of a Canonical Equivalence 4
   Capturing the formal essence of dynamic algorithms on “recursive” decom-
   positions: parse trees and Myhill-Nerode type congruences.

3 Measuring Graphs: Clique-width and Rank-width 8
   Measuring tree-likeness of a graph: the */-widths.

4 #SAT – our Sample Application 14
   Giving an FPT algorithm which is single-exponential in the rank-width.

5 Final remarks 19
1 Decomposing the Input and running Dynamic Algorithms

- A typical idea for a dynamic algorithm on a recursive decomposition:
  - Capture all relevant information 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.
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. . .

• 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.

• 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.

• 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) $\leftrightarrow$
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.

• 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) $\leftrightarrow$ 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 concatenation 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 concatenation 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 concatenation 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:

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

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}.$$
2 The Concept of a Canonical Equivalence

How does the right congruence extend from formal words with the concatenation 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:

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

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

- 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 actual meaning of “boundary” and “join”).
Definition. The *canonical equivalence* of $\mathcal{P}$ on the universe $\mathcal{U}_k$ is defined:

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

$$G_1 \otimes H \in \mathcal{P} \iff G_2 \otimes H \in \mathcal{P}$$
Decision properties, or more?

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

$G_1 \approx_{\mathcal{P},k} G_2$ 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:

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

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

- 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 $\varphi$. 
Decision properties, or more?

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

$$G_1 \approx_{\mathcal{P},k} G_2 \text{ for any } G_1, G_2 \in U_k \text{ if and only if, for all } H \in U_k,$$

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

• 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 $\varphi$.

---

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

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

$$(G_1, \varphi_1) \otimes (H, \varphi) \models \mathcal{P} \iff (G_2, \varphi_2) \otimes (H, \varphi) \models \mathcal{P}$$
Decision properties, or more?

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

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

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

- 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 $\varphi$.

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

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

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

- For simplicity, solution fragments $\varphi$ 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:

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

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

- 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 $\varphi$.

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

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

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

- For simplicity, solution fragments $\varphi$ 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.
Some particular issues, beyond Myhill-Nerode

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

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

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

- Are the elements of $\mathcal{U}_k$ required *recursively decomposable*?
Some particular issues, beyond Myhill-Nerode

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

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

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

- Are the elements of $\mathcal{U}_k$ required *recursively decomposable*?
  - somehow surprisingly, does not seem to play role. . .
Some particular issues, beyond Myhill-Nerode

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

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

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

- Are the elements of $\mathcal{U}_k$ required recursively decomposable?
  - somehow surprisingly, does not seem to play role.

- Can we have a different “right-hand-side universe” $H \in \mathcal{U}'_k$?
  - yes, useful e.g. for bi-rank-width of digraphs.
Some particular issues, beyond Myhill-Nerode

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

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

• Are the elements of $\mathcal{U}_k$ required recursively decomposable?
  – somehow surprisingly, does not seem to play role. . .

• Can we have a different “right-hand-side universe” $H \in \mathcal{U}_k'$?
  – yes, useful e.g. for bi-rank-width of digraphs.

• Can we use more different join operators $\otimes$? Why?
  – related to “prepartitioning” (expectation) of right-hand universe.
Some particular issues, beyond Myhill-Nerode

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

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

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

- Are the elements of $\mathcal{U}_k$ required recursively decomposable?
  - somehow surprisingly, does not seem to play role.

- Can we have a different "right-hand-side universe" $H \in \mathcal{U}_k'$?
  - yes, useful e.g. for bi-rank-width of digraphs.

- Can we use more different join operators $\otimes$? Why?
  - related to "prepartitioning" (expectation) of right-hand universe.

- **XP algorithms**, i.e. getting away from finite automata?
Some particular issues, beyond Myhill-Nerode

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

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

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

- Are the elements of $\mathcal{U}_k$ required recursively decomposable?  
  – somehow surprisingly, does not seem to play role. . .

- Can we have a different “right-hand-side universe” $H \in \mathcal{U}_k'$?  
  – yes, useful e.g. for bi-rank-width of digraphs.

- Can we use more different join operators $\otimes$? Why?  
  – related to “prepartitioning” (expectation) of right-hand universe.

- XP algorithms, i.e. getting away from finite automata?  
  – yes, still works quite nicely, cf. [Ganian, PH, Obdržálek, 09].  
  – brings new application issues such as “quantification inside $\otimes$” (cf. sol. fragments), or a “second-level” congruence on top of $\approx_{\mathcal{P},k}$.  

P. Hliněný, PCCR 2010, Brno CZ 6 Myhill–Nerode Meets Parameterized...
Parse trees of decompositions

To give an algor. 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

To give an algor. usable meaning to the 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* $\otimes$ $\leftrightarrow$ the way pieces of $G$ “stick together” in decomp.
Parse trees of decompositions

To give an algor. usable meaning to the 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 $k$** ↔ restricted bag-size / width / etc in decomposition
  - **join operator $\otimes$** ↔ the way pieces of $G$ “stick together” in decomp.

- This can be (visually) seen as…
3 Measuring Graphs: Clique-width and Rank-width

Motivation: Trees are easy to understand and to handle, so how “tree-like” our graph is in some well-defined sense (the width)?

- A topic occurring both in pure theory (e.g. Graph Minors), and in algorithms (Fixed parameter tractability).
3 Measuring Graphs: Clique-width and Rank-width

Motivation: Trees are easy to understand and to handle, so how “tree-like” our graph is in some well-defined sense (the width)?

- A topic occurring 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 ...
3 Measuring Graphs: Clique-width and Rank-width

Motivation: Trees are easy to understand and to handle, so how “tree-like” our graph is in some well-defined sense (the \textit{width})?

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

- Many definitions known, e.g. \textit{tree-width}, \textit{path-width}, \textit{branch-width}, \textit{DAG-width} . . .

- \textbf{Clique-width} – another graph complexity measure [Courcelle and Olariu], defined by operations on \textit{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 \(j\).
3 Measuring Graphs: Clique-width and Rank-width

Motivation: Trees are easy to understand and to handle, so how “tree-like” our graph is in some well-defined sense (the width)?

- A topic occurring 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 $j$.

  \[ \rightarrow \text{ giving the expression tree (parse tree) for clique-width.} \]
### Rank-decomposition

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

\[
\varrho_G(X) = \text{rank of } X \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \mod 2
\]
**Rank-decomposition**

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

  \[
  \varrho_G(X) = \text{rank of } X \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \mod 2
  \]

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

- \[
  \text{width}(e) = \varrho_G(X) \text{ where } X \text{ is displayed by } f \text{ in the tree.}
  \]
Rank-decomposition

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

$$\varrho_G(X) = \text{rank of } \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix} \mod 2$$

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

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

- \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 $t \leq k \leq 2^{t+1} - 1$.
- Both these measures are $NP$-hard in general.
Comparing these two

- Rank-width $t$ is related to clique-width $k$ as $t \leq k \leq 2^{t+1} - 1$.
- 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 \( t \leq k \leq 2^{t+1} - 1 \).

- 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 $t \leq k \leq 2^{t+1} - 1$.

• 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 width-$t$ 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 $t \leq k \leq 2^{t+1} - 1$.
- 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 width-$t$ rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.
- And new results show that certain algorithms designed on rank-decompositions run faster than their analogues designed on clique-width expressions... (subst. $poly(t)$ in place of $cw$, instead of $2^t$)
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 rank.
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 rank.

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

- **Bilinear product** approach of [Courcelle and Kanté, 07]:

  - boundary \( \sim \) labeling \( \text{lab} : V(G) \rightarrow 2^{\{1,2,\ldots,t\}} \) (multi-colouring),
  - join \( \sim \) bilinear form \( g \) over \( GF(2)^t \) (i.e. "odd intersection") s.t.
    \[
    \text{edge } uv \iff \text{lab}(u) \cdot g \cdot \text{lab}(v) = 1.
    \]
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 rank.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  
  - **boundary** ~ labeling $lab : V(G) \rightarrow 2^{\{1,2,\ldots,t\}}$ (multi-colouring),
  - **join** ~ bilinear form $g$ over $GF(2)^t$ (i.e. “odd intersection”) s.t.
    
    $$
    \text{edge } uv \leftrightarrow lab(u) \cdot g \cdot lab(v) = 1.
    $$

- Join → a composition operator with relabelings $f_1, f_2$;
  
  $$(G_1, lab^1) \otimes [g | f_1, f_2] (G_2, lab^2) = (H, lab)$$

  $\implies$ 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 rank.

• **Bilinear product** approach of [Courcelle and Kanté, 07]:
  
  – *boundary* \(\sim\) labeling \(\text{lab} : V(G) \rightarrow 2^{\{1,2,\ldots,t\}}\) (multi-colouring),
  
  – *join* \(\sim\) bilinear form \(g\) over \(GF(2)^t\) (i.e. “odd intersection”) s.t.
    
    \[\text{edge } uv \iff \text{lab}(u) \cdot g \cdot \text{lab}(v) = 1.\]

• Join \(\rightarrow\) a composition operator with relabelings \(f_1, f_2\);

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

  \(\implies\) the rank-width **parse tree** [Ganian and PH, 08]:

  \(t\)-labeling parse tree for \(G\) \(\iff\) rank-width of \(G \leq t.\)
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 rank.

- **Bilinear product** approach of [Courcelle and Kanté, 07]:
  - boundary ~ labeling \( \text{lab} : V(G) \to 2^{\{1,2,\ldots,t\}} \) (multi-colouring),
  - join ~ bilinear form \( g \) over \( GF(2)^t \) (i.e. “odd intersection”) s.t. \( \text{edge } uv \leftrightarrow \text{lab}(u) \cdot g \cdot \text{lab}(v) = 1. \)

- Join \( \to \) a composition operator with relabelings \( f_1, f_2; \)
  \[
  (G_1, \text{lab}^1) \otimes [g \mid f_1, f_2] \ (G_2, \text{lab}^2) = (H, \text{lab})
  \]

  \( \implies \) the rank-width parse tree [Ganian and PH, 08]:
  - \( t \)-labeling parse tree for \( G \) \iff rank-width of \( G \leq t. \)

- 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):

\[
\begin{array}{c}
\otimes[id | \cdot, \cdot] \\
\otimes[id | id, 1 \rightarrow \emptyset] \\
\otimes[id | id, 1 \rightarrow 2] \\
\otimes[id | 1 \rightarrow 2, id] \\
\end{array}
\]

\[
\begin{array}{c}
\circ a \\
\circ b \\
\circ c \\
\circ d \\
\circ e \\
\end{array}
\]

\[
\begin{array}{c}
d \{1\} \\
e \{1\} \\
b \{1\} \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
c \{1\} \\
e \{1\} \\
a \{1\} \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
d \{2\} \\
e \{1\} \\
a \{1\} \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
d \{2\} \\
e \{1\} \\
a \{1\} \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
c \{2\} \\
e \{1\} \\
a \{1\} \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
d \\
\rightarrow \\
\end{array}
\]

\[
\begin{array}{c}
C_5 \\
\end{array}
\]
4 \textbf{#SAT – our Sample Application}

- \textit{#SAT} – counting satisfying assignments of a CNF formula, a well-known \#P-hard problem.
4 #SAT – our Sample Application

- **#SAT** – counting satisfying assignments of a CNF formula, a well-known \#P-hard problem.

- FPT solutions on *formulas of bounded *-width*:
  - [Fisher, Makowsky, and Ravve, 08] – tree-width and clique-width,
  - [Samer and Szeider, 10] – tree-width improved.
4 #SAT – our Sample Application

- **#SAT** – counting satisfying assignments of a CNF formula, a well-known \#P-hard problem.

- FPT solutions on *formulas of bounded *-width*:
  - [Fisher, Makowsky, and Ravve, 08] – tree-width and clique-width,
  - [Samer and Szeider, 10] – tree-width improved.

- On the other hand...

**Quote.** [Samer and Szeider, 10] – regarding #SAT and *clique-width*:

... A single-exponential algorithm (for #SAT) is due to Fisher, Makowsky, and Ravve. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error and are of limited practical value.
4 \textbf{#SAT – our Sample Application}

- \textbf{#SAT} – counting satisfying assignments of a CNF formula, a well-known \#P-hard problem.

- FPT solutions on \textit{formulas of bounded \*-*width}:
  - [Fisher, Makowsky, and Ravve, 08] – tree-width and clique-width,
  - [Samer and Szeider, 10] – tree-width improved.

- On the other hand. . .

\textbf{Quote.} [Samer and Szeider, 10] – regarding \#SAT and \textit{clique-width}:

\ldots\ A single-exponential algorithm (for \#SAT) is due to Fisher, Makowsky, and Ravve. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error and are of limited practical value.

\textbf{Where is the problem?}

A resulting \textbf{double-exponential} worst-case dependency on a width estimate!
The problem, again

**Quote.** [Samer and Szeider, 10] – regarding #SAT and *clique-width*:

A single-exponential algorithm (for #SAT) is due to Fisher, Makowsky, and Ravve. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error and are of limited practical value.

**Our answer** – considering *rank-width*:
The problem, again

**Quote.** [Samer and Szeider, 10] – regarding \#SAT and *clique-width*:

A single-exponential algorithm (for \#SAT) is due to Fisher, Makowsky, and Ravve. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error and are of limited practical value.

**Our answer** – considering *rank-width*:

- No loss in the promised width, and yet single-exponential in it.
The problem, again

Quote. [Samer and Szeider, 10] – regarding \#SAT and clique-width:

A single-exponential algorithm (for \#SAT) is due to Fisher, Makowsky, and Ravve. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error and are of limited practical value.

Our answer – considering rank-width:

• No loss in the promised width, and yet single-exponential in it.
• A clear and rigorous algorithm employing many of the above tricks.

Theorem. [Ganian, PH, Obdržálek, 10] \#SAT solved in FPT time

\[ O(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|) \]

where \( t \) is the signed rank-width of the input instance (CNF formula) \( \phi \).
Signed graphs of CNF formulas

- The common way to measure structure / width of a formula:

  \[
  \text{vertices} \ := \ V \cup C \quad \text{variables and clauses of } \phi.
  \]
Signed graphs of CNF formulas

- The common way to measure structure / width of a formula:

  **vertices** := $V \cup C$  variables and clauses of $\phi$.

  **edges** := $E^+ \cup E^-$  where

  \[ x_i c_j \in E^+ \text{ if } c_j = (\cdots \lor x_i \ldots) \in C, \text{ and} \]
  \[ x_i c_j \in E^- \text{ if } c_j = (\cdots \lor \neg x_i \ldots) \in C. \]
Signed graphs of CNF formulas

- The common way to measure structure / width of a formula:

  vertices \( := V \cup C \) \( \) variables and clauses of \( \phi \).

  edges \( := E^+ \cup E^- \) where

  \( x_i c_j \in E^+ \) if \( c_j = (\cdots \lor x_i \cdots) \in C \), and

  \( x_i c_j \in E^- \) if \( c_j = (\cdots \lor \neg x_i \cdots) \in C \).

- Signed clique-width – using distinct operations for \( E^+ \) and \( E^- \) (ordinary clique-width is not enough!).
Signed graphs of CNF formulas

• The common way to measure structure/width of a formula:

  vertices := $V \cup C$ variables and clauses of $\phi$.

  edges := $E^+ \cup E^-$ where

  $x_i c_j \in E^+$ if $c_j = (\cdots \lor x_i \cdots) \in C$, and
  $x_i c_j \in E^-$ if $c_j = (\cdots \lor \neg x_i \cdots) \in C$.

• Signed clique-width – using distinct operations for $E^+$ and $E^-$
  (ordinary clique-width is not enough!).

• Signed rank-width – using separate joins for $E^+$ and $E^-$, formally
  $G = G^+ \cup G^-$ on the same vertex set (sim. bi-rank-width).
Signed graphs of CNF formulas

The common way to measure structure/width of a formula:

- **vertices**: $V \cup C$ variables and clauses of $\phi$.
- **edges**: $E^+ \cup E^-$ where
  
  $x_i c_j \in E^+$ if $c_j = (\cdots \lor x_i \ldots) \in C$, and
  $x_i c_j \in E^-$ if $c_j = (\cdots \lor \neg x_i \ldots) \in C$.

- **Signed clique-width** – using distinct operations for $E^+$ and $E^-$ (ordinary clique-width is not enough!).

- **Signed rank-width** – using separate joins for $E^+$ and $E^-$, formally
  
  $G = G^+ \cup G^-$ on the same vertex set (sim. bi-rank-width).

Then

$$G_1 \oplus G_2 = (G_1^+ \oplus G_2^+) \cup (G_1^- \oplus G_2^-)$$

and the same decomposition is used.
The canonical equivalence for SAT

- Corresp. $G = G[\phi]$ signed graph $\leftrightarrow \phi = \phi[G]$ CNF formula.
The canonical equivalence for SAT

• Corresp. \( G = G[\phi] \) signed graph \( \iff \phi = \phi[G] \) CNF formula.

• Valuation \( \nu_G : V \to \{0, 1\} \).
The canonical equivalence for SAT

- Corresp. \( G = G[\phi] \) signed graph \( \iff \phi = \phi[G] \) CNF formula.
- Valuation \( \nu_G : V \to \{0, 1\} \).
- The canonical equivalence: \( (G_1, \nu_1) \approx_{SAT,t} (G_2, \nu_2) \) iff, for all \((H, \nu_H)\),

\[
\nu_1 \cup \nu_H \models \phi[G_1 \otimes H] \iff \nu_2 \cup \nu_H \models \phi[G_2 \otimes H].
\]
The canonical equivalence for SAT

• Corresp. \( G = G[\phi] \) signed graph \( \iff \phi = \phi[G] \) CNF formula.

• Valuation \( \nu_G : V \to \{0, 1\} \).

• The canonical equivalence: \((G_1, \nu_1) \approx_{SAT,t} (G_2, \nu_2)\) iff, for all \((H, \nu_H)\),

\[
\nu_1 \cup \nu_H \models \phi[G_1 \otimes H] \iff \nu_2 \cup \nu_H \models \phi[G_2 \otimes H].
\]

Proposition. \((G_1, \nu_1) \approx_{SAT,t} (G_2, \nu_2)\) if the foll. equal for \((G_i, \nu_i), i = 1, 2:\)

– the set of \(G_i^+\)-labels occuring at true (under \(\nu_i\)) variables,
The canonical equivalence for SAT

- Corresp. $G = G[\phi]$ signed graph $\iff \phi = \phi[G]$ CNF formula.

- Valuation $\nu_G : V \to \{0, 1\}$.

- The canonical equivalence: $(G_1, \nu_1) \approx_{SAT,t} (G_2, \nu_2)$ iff, for all $(H, \nu_H)$,

  $$\nu_1 \cup \nu_H \models \phi[G_1 \otimes H] \iff \nu_2 \cup \nu_H \models \phi[G_2 \otimes H].$$

Proposition. $(G_1, \nu_1) \approx_{SAT,t} (G_2, \nu_2)$ if the foll. equal for $(G_i, \nu_i)$, $i = 1, 2$:

- the set of $G_i^+$-labels occurring at true (under $\nu_i$) variables,
- analog., the set of $G_i^-$-labels of false (under $\nu_i$) variables, and
The canonical equivalence for SAT

- Corresp. \( G = G[\phi] \) signed graph \( \iff \phi = \phi[G] \) CNF formula.

- Valuation \( \nu_G : V \to \{0, 1\} \).

- The canonical equivalence: \((G_1, \nu_1) \approx_{\text{SAT}, t} (G_2, \nu_2)\) iff, for all \((H, \nu_H)\),
  \[
  \nu_1 \cup \nu_H \models \phi[G_1 \otimes H] \iff \nu_2 \cup \nu_H \models \phi[G_2 \otimes H].
  \]

Proposition. \( (G_1, \nu_1) \approx_{\text{SAT}, t} (G_2, \nu_2) \) if the foll. equal for \((G_i, \nu_i)\), \(i = 1, 2\):

- the set of \( G^+_i \)-labels occurring at true (under \( \nu_i \)) variables,
- analog., the set of \( G^-_i \)-labels of false (under \( \nu_i \)) variables, and
- the set of pair labels of all unsatisfied (under \( \nu_i \)) clauses of \( \phi[G_i] \).
The canonical equivalence for SAT

- Corresp. \( G = G[\phi] \) signed graph \( \iff \phi = \phi[G] \) CNF formula.

- Valuation \( \nu_G : V \to \{0, 1\} \).

- The canonical equivalence: \((G_1, \nu_1) \approx_{\text{SAT}, t} (G_2, \nu_2)\) iff, for all \((H, \nu_H)\),
  \[
  \nu_1 \cup \nu_H \models \phi[G_1 \otimes H] \iff \nu_2 \cup \nu_H \models \phi[G_2 \otimes H].
  \]

**Proposition.** \((G_1, \nu_1) \approx_{\text{SAT}, t} (G_2, \nu_2)\) if the foll. equal for \((G_i, \nu_i)\), \(i = 1, 2\):

- the set of \(G_i^+\)-labels occurring at true (under \(\nu_i\)) variables,
- analog., the set of \(G_i^-\)-labels of false (under \(\nu_i\)) variables, and
- the set of pair labels of all unsatisfied (under \(\nu_i\)) clauses of \(\phi[G_i]\).

Easy to prove..., but does it help?

Subsets of labels from \(2^{\{1,2,\ldots,t\}}\) \(\longrightarrow\) \(\Omega(2^{2^t})\) classes!
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:

- **Linear algebra:**
  
  Subset of labels $\rightarrow$ the \emph{spanning subspace} in $GF(2)^t$. 
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:

- **Linear algebra:**
  Subset of labels $\rightarrow$ the *spanning subspace* in $GF(2)^t$.

**Theorem.** [Goldman and Rota, 69] The number of subspaces of $GF(2)^t$ is

$$S(t) \leq 2^{t(t+1)/4} \text{ for all } t \geq 12.$$
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:

- **Linear algebra:**
  Subset of labels $\rightarrow$ the *spanning subspace* in $GF(2)^t$.

**Theorem.** [Goldman and Rota, 69] The number of subspaces of $GF(2)^t$ is

$$S(t) \leq 2^{t(t+1)/4} \text{ for all } t \geq 12.$$

- **Expectation:**
  Labels of unsat. clauses $\rightarrow$ *expected labels* of variables in $H$,
  and the subspace trick once again.
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:

- **Linear algebra**:  
  Subset of labels $\rightarrow$ the *spanning subspace* in $GF(2)^t$.

**Theorem.** [Goldman and Rota, 69] The number of subspaces of $GF(2)^t$ is  
$$S(t) \leq 2^{t(t+1)/4}$$  
for all $t \geq 12$.

- **Expectation**:  
  Labels of unsat. clauses $\rightarrow$ *expected labels* of variables in $H$,  
  and the subspace trick once again.

In other words, $\approx_{SAT,t}$ “suitably restricted” to $(H, \nu_H)$’s of the expected label subspaces of its false and true variables. . .
Getting coarser equivalences for SAT

We improve the runtime with the following two main tricks:

- **Linear algebra:**
  
  Subset of labels $\rightarrow$ the *spanning subspace* in $GF(2)^t$.

  **Theorem.** [Goldman and Rota, 69] The number of subspaces of $GF(2)^t$ is
  
  $$S(t) \leq 2^{t(t+1)/4}$$
  
  for all $t \geq 12$.

- **Expectation:**
  
  Labels of unsat. clauses $\rightarrow$ *expected labels* of variables in $H$,
  
  and the subspace trick once again.

  In other words, $\approx_{SAT,t}$ “suitably restricted” to $(H, \nu_H)$’s of the *expected*
  label subspaces of its false and true variables.

**Conclusion.** Breaking the satisfying assignments of $\phi$ into $S(t)^4$ classes,

and processing a node of the parse tree in $O^*(S(t)^6)$.
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:

- The use of *Myhill–Nerode type congruences* in dynamic progr. alg. design
  
  - can give very rigorous proofs for algorithms (almost for free), and
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:

- The use of *Myhill–Nerode type congruences* in dynamic progr. alg. design
  - can give very rigorous proofs for algorithms (almost for free), and
  - immediately provides a rather simple test of “what is possible”.
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:

- The use of *Myhill–Nerode type congruences* in dynamic progr. alg. design
  - can give very *rigorous proofs* for algorithms (almost for free), and
  - immediately provides a rather simple test of “*what is possible*”.

- *Rank-width* to be used in place of *clique-width* in param. algorithms.
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:

- The use of Myhill–Nerode type congruences in dynamic progr. alg. design
  - can give very rigorous proofs for algorithms (almost for free), and
  - immediately provides a rather simple test of “what is possible”.

- Rank-width to be used in place of clique-width in param. algorithms.

- Rank-width is useful for variants of SAT via the signed graph.
5 Final remarks

Our talk suggests (tries to, at least) the following research directions... as ordered from the very general one to the very concrete example:

- The use of *Myhill–Nerode type congruences* in dynamic progr. alg. design
  - can give very *rigorous proofs* for algorithms (almost for free), and
  - immediately provides a rather simple test of “what is possible”.

- *Rank-width* to be used in place of *clique-width* in param. algorithms.

- Rank-width is useful for variants of *SAT* via the *signed graph*.

THANK YOU FOR YOUR ATTENTION