Unified Approach to Polynomial Algorithms on Graphs of Bounded (bi-)Rank-width

Petr Hliněný,∗
Robert Ganian and Jan Obdržálek

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

e-mail: hlineny@fi.muni.cz
ganian@mail.muni.cz
obdrzalek@fi.muni.cz
http://www.fi.muni.cz/~hlineny
0 Introduction

In this presentation, we will mix some very general (and abstract) ideas about graph “width” decompositions and dynamic programming algorithms on those, with specific applications to efficient algorithms for hard problems running on rank-decompositions of graphs.
# 0 Introduction

In this presentation, we will mix some very general (and abstract) ideas about graph “width” decompositions and dynamic programming algorithms on those, with specific applications to efficient algorithms for hard problems running on rank-decompositions of graphs.

## Talk Outline

| 1 Measuring Graph “Width” | 3 |
| 2 Dynamic Algorithms and Parse Trees | 7 |
| 3 Parse Trees for Rank-Decompositions | 10 |
| 4 Canonical Equivalence and Algorithms | 12 |
| 5 Unified Design Style of XP Algorithms | 14 |
| 6 Conclusions | 17 |
1 Measuring Graph “Width”

Motivation: Trees are easy to understand and to handle, so how “tree-like” our graphs are . . . , in some well-defined sense?

- A topic occurring 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 graphs are . . . , in some well-defined sense?

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

• Many definitions have been studied so far, 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 graphs are ..., in some well-defined sense?

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

- Many definitions have been studied so far, e.g. \textit{tree-width}, \textit{path-width}, \textit{branch-width}, \textit{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-Decompositions (a “better view” of clique-width)

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

\[
\varrho_G(X) = \text{rank of } X \begin{pmatrix}
0 & 1 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 1
\end{pmatrix} \text{ modulo } 2
\]
**Rank-Decompositions** (a “better view” of clique-width)

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

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

**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-Decompositions** (a “better view” of clique-width)

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

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

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

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

**Rank-width** = \( \min_{\text{rank-dec. of } G} \max \{ \text{width}(f) : f \text{ tree edge} \} \)
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 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 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 limited polynomial size, ideally even constant independent of the input 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 *limited polynomial size*, ideally even constant independent of the input 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 limited polynomial size, ideally even constant independent of the input 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) ↔ right congruence classes on the words (of a regular language).
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 limited polynomial size, ideally even constant independent of the input 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) ↔ 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 concatenation 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 concatenation operation to, say, graphs with a kind of a “join” operation?

- Consider the universe of graphs $U_k$ 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 concatenation 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:

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

$$G_1 \oplus H \in \mathcal{P} \iff G_2 \oplus H \in \mathcal{P}.$$
The concept of a canonical equivalence

How does the right congruence extend from formal words with the concatenation operation to, say, graphs with a kind of a “join” operation?

• Consider the universe of graphs $U_k$ 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 $U_k$ is defined:

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

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

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

- Join → a composition operator with relabelings \( f_1, f_2 \);
  
  \[
  (G_1, \text{lab}^1) \otimes [g | f_1, f_2] (G_2, \text{lab}^2) = (H, \text{lab})
  \]
  
  \( \implies \) 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 $lab : V(G) \rightarrow 2^{\{1,2,\ldots,t\}}$ (multi-colouring),
  - join $\sim$ bilinear form $g$ over $GF(2)^t$ (i.e. “odd intersection”) s.t.
    $$\text{edge } uv \leftrightarrow lab(u) \cdot g \cdot lab(v) = 1.$$

- Join $\rightarrow$ a **composition** operator with relabelings $f_1, f_2$;
  $$(G_1, lab^1) \otimes [g \mid f_1, f_2] (G_2, lab^2) = (H, lab)$$
  $\implies$ the rank-width **parse tree** [Ganian and PH, 08]:
  - $t$-labeling parse tree for $G$ $\iff$ rank-width of $G \leq t$. 
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 \( \text{lab} : V(G) \rightarrow 2^{\{1,2,...,t\}} \) (multi-colouring),
  - join \( \sim \) 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 \( \rightarrow \) a composition operator with relabelings \( f_1, f_2 \);
  \[ (G_1, \text{lab}^1) \otimes [g \mid f_1, f_2] (G_2, \text{lab}^2) = (H, \text{lab}) \]
  \( \Longrightarrow \) 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].
Parse tree. An example generating the cycle $C_5$ (of rank-width 2):

\[
\begin{array}{c}
\otimes[id | \cdot, \cdot] \\
\otimes[id | id, 1 \rightarrow \emptyset] \\
\otimes[id | id, 1 \rightarrow 2] \\
\otimes[id | 1 \rightarrow 2, id] \\
\circ a \\
\circ b \\
\circ c \\
\circ d \\
\circ e \\
\end{array}
\]
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])
\(P\) is accepted by a *finite tree automaton* on parse trees of boundary size \(\leq k\) \(\iff\) the canonical equivalence \(\approx_{P,k}\) has finitely many classes on \(U_k\).

(Actually, this is a “metatheorem” which requires several more unspoken technical conditions on the parse trees to hold true...)
Extended canonical equivalence

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

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

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

- 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 \( \phi \) 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

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

- 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 \( \phi \) 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 FPT time \( O(f(t) \cdot n) \).

Core idea: In dynamic processing of the given parse tree, record optimal representatives of each class of the extended canonical equivalence \( \approx_{\phi,t} \) ...
5 Unified Design Style of XP Algorithms

(XP: running in time $O(n^{f(k)})$, not FPT)

Starting point: For many problems $\mathcal{P}$, the number of classes of $\cong_{\mathcal{P},k}$ depends on the input size $n$ (→ likely no FPT algorithm exists).
5 Unified Design Style of XP Algorithms

(XP: running in time $O(n^{f(k)})$, not FPT)

**Starting point:** For many problems $\mathcal{P}$, the number of classes of $\approx_{\mathcal{P},k}$ depends on the input size $n$ (→ likely no FPT algorithm exists).

Yet there are known algorithms for them dynamically processing “information” of polynomial size $O(n^{f(k)})$... How do they work?

We try to give a unified formal description...
5 Unified Design Style of XP Algorithms

(XP: running in time $O(n^{f(k)})$, not FPT)

**Starting point:** For many problems $\mathcal{P}$, the number of classes of $\approx_{\mathcal{P},k}$ depends on the input size $n$ (→ likely no FPT algorithm exists).

Yet there are known algorithms for them dynamically processing “information” of polynomial size $O(n^{f(k)})$... How do they work?

We try to give a unified formal description...

**In our parse-tree (width $k$) formalism**, we

- assoc. the equiv. classes of $\approx_{\mathcal{P},k}$ with an enum. of suitable “fragments”,

P. Hliněný et al., CSASC 2010  14  XP algorithms for bounded (bi-)rank-width
5 Unified Design Style of XP Algorithms

(XP: running in time $O(n^{f(k)})$, not FPT)

Starting point: For many problems $\mathcal{P}$, the number of classes of $\approx_{\mathcal{P},k}$ depends on the input size $n$ (→ likely no FPT algorithm exists).

Yet there are known algorithms for them dynamically processing “information” of polynomial size $O(n^{f(k)})$... How do they work?

We try to give a unified formal description...

In our parse-tree (width $k$) formalism, we

- assoc. the equiv. classes of $\approx_{\mathcal{P},k}$ with an enum. of suitable “fragments”,
- where the number of distinct “fragments” depends only on $k$, 
5 Unified Design Style of XP Algorithms

(\(\text{XP: running in time } O(n^{f(k)}), \text{ not FPT}\))

**Starting point:** For many problems \(P\), the number of classes of \(\approx P, k\) depends on the input size \(n\) (\(\rightarrow \) likely no FPT algorithm exists).

Yet there are known algorithms for them dynamically processing “information” of polynomial size \(O(n^{f(k)})\). . . How do they work?

We try to give a unified formal description . . .

**In our parse-tree (width \(k\)) formalism,** we

- assoc. the equiv. classes of \(\approx P, k\) with an enum. of suitable “fragments”,
- where the number of distinct “fragments” depends only on \(k\),
- and we can recombine the fragment enumerations efficiently.
Example 1: Hamiltonian path

XP algorithm wrt. clique-width given by [Espelage, Gurski, and Wanke, 2001].

**Theorem.** Decide whether a graph $G$ of rank-width $t$ has a *Hamiltonian path* in time

$$O \left( |V(G)|^{\ell(t)} \right)$$

where $\ell(t) = 4^{t+1} + O(1)$
Example 1: Hamiltonian path

XP algorithm wrt. clique-width given by [Espelage, Gurski, and Wanke, 2001].

**Theorem.** Decide whether a graph $G$ of rank-width $t$ has a *Hamiltonian path* in time

$$O\left(|V(G)|^{\ell(t)}\right)$$

where $\ell(t) = 4^{t+1} + O(1)$.

**Proof:**

- Considering a *Hamiltonian path* $P$ in the join $G \oplus H$,
**Example 1: Hamiltonian path**

XP algorithm wrt. clique-width given by [Espelage, Gurski, and Wanke, 2001].

**Theorem.** Decide whether a graph $G$ of rank-width $t$ has a Hamiltonian path in time

\[ O \left( |V(G)|^{\ell(t)} \right) \text{ where } \ell(t) = 4^{t+1} + O(1). \]

**Proof:**

- Considering a Hamiltonian path $P$ in the join $G \oplus H$,
- the “fragments” are the subpaths $P_i \subseteq P$ on the $G$-side
Example 1: Hamiltonian path

XP algorithm wrt. clique-width given by [Espelage, Gurski, and Wanke, 2001].

**Theorem.** Decide whether a graph $G$ of rank-width $t$ has a Hamiltonian path in time

$$O \left( |V(G)|^{\ell(t)} \right)$$

where $\ell(t) = 4^{t+1} + O(1)$.

**Proof:**

- Considering a Hamiltonian path $P$ in the join $G \oplus H$,
- the “fragments” are the subpaths $P_i \subseteq P$ on the $G$-side
  - identified by labeling pairs of their ends (only $4^t$ distinct!),
  - and enumerated at every parse tree node as one multiset.
**Example 1: Hamiltonian path**

XP algorithm wrt. clique-width given by [Espelage, Gurski, and Wanke, 2001].

**Theorem.** Decide whether a graph $G$ of rank-width $t$ has a Hamiltonian path in time

$$O \left( |V(G)|^{\ell(t)} \right)$$

where $\ell(t) = 4^{t+1} + O(1)$.

**Proof:**

- Considering a Hamiltonian path $P$ in the join $G \oplus H$,
- the “fragments” are the subpaths $P_i \subseteq P$ on the $G$-side
  - identified by labeling pairs of their ends (only $4^t$ distinct!),
  - and enumerated at every parse tree node as one multiset.
- Straightforward dynamic alg. processing then gives the result.
Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).
Example 2: Defective colouring

**Defective** \((\ell, q)\)-**colouring** – partition the vertices into \(\ell\) parts such that

– each part induces a subgraph of max. degree \(\leq q\).

Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).


**Fact.** For fixed \(q\), this is an *MSOL partitioning problem*. 
Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).


Fact. For fixed \(q\), this is an MSOL partitioning problem.

Theorem. The defective \((\ell, q)\)-colouring problem with fixed \(\ell\) parts (i.e. minimizing \(q\)) can be solved on a graph \(G\) of rank-width \(t\) in time

\[
O\left(|V(G)|^{k(t,\ell)}\right) \text{ where } k(t, \ell) = 4\ell \cdot 2^t + O(1)
\]
Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).


**Fact.** For fixed \(q\), this is an **MSOL partitioning problem**.

**Theorem.** The defective \((\ell, q)\)-colouring problem with fixed \(\ell\) parts (i.e. minimizing \(q\)) can be solved on a graph \(G\) of rank-width \(t\) in time

\[
O\left(|V(G)|^{k(t,\ell)}\right) \quad \text{where} \quad k(t, \ell) = 4\ell \cdot 2^t + O(1).
\]

**Proof:**

- Consider separately each one colour class \(X\).
Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).


**Fact.** For fixed \(q\), this is an MSOL partitioning problem.

**Theorem.** The defective \((\ell, q)\)-colouring problem with fixed \(\ell\) parts (i.e. minimizing \(q\)) can be solved on a graph \(G\) of rank-width \(t\) in time
\[
O \left( |V(G)|^{k(t, \ell)} \right)
\]
where \(k(t, \ell) = 4\ell \cdot 2^t + O(1)\).

**Proof:**

- Consider separately each one colour class \(X\).
- A “fragment” – one vertex labeling in \(X\), but one needs also to record its max. degree in \(X\)!
Example 2: Defective colouring

Defective \((\ell, q)\)-colouring – partition the vertices into \(\ell\) parts such that
– each part induces a subgr. of max. degree \(\leq q\).


Fact. For fixed \(q\), this is an MSOL partitioning problem.

Theorem. The defective \((\ell, q)\)-colouring problem with fixed \(\ell\) parts (i.e. minimizing \(q\)) can be solved on a graph \(G\) of rank-width \(t\) in time

\[
O \left( |V(G)|^{k(t, \ell)} \right)
\]

where \(k(t, \ell) = 4\ell \cdot 2^t + O(1)\).

Proof:

- Consider separately each one colour class \(X\).
- A “fragment” – one vertex labeling in \(X\), but one needs also to record its max. degree in \(X\)!
- Slightly out of our formalism, and so deserves a closer look…
6 Conclusions

• The power of the *Myhill–Nerode–type* formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?
6 Conclusions

• The power of the Myhill–Nerode–type formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?

• Still, one would like to see an explicit (perhaps logic-based) framework for XP algorithms, analogously to the MSOL framework with FPT algorithms, cf. [Courcelle] et al.
6 Conclusions

• The power of the Myhill–Nerode–type formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?

• Still, one would like to see an explicit (perhaps logic-based) framework for XP algorithms, analogously to the MSOL framework with FPT algorithms, cf. [Courcelle] et al.
  
  – Our presented unified approach shows this should be possible. . .
6 Conclusions

• The power of the *Mynhill–Nerode–type* formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?

• Still, one would like to see an explicit (perhaps logic-based) framework for XP algorithms, analogously to the MSOL framework with FPT algorithms, cf. [Courcelle] et al.

  – Our presented unified approach shows this should be possible...

  – And very recently, [Král’, Obdržálek, and PH, 2010] have succeeded in finding such a framework.
6 Conclusions

• The power of the Myhill–Nerode–type formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?

• Still, one would like to see an explicit (perhaps logic-based) framework for XP algorithms, analogously to the MSOL framework with FPT algorithms, cf. [Courcelle] et al.
  
  – Our presented unified approach shows this should be possible. . .
  
  – And very recently, [Král’, Obdržálek, and PH, 2010] have succeeded in finding such a framework.

• BTW (totally unrelated. . .)

  Have you already heard that the crossing number of almost planar graphs is NP-complete? [Cabello and Mohar, 2010]
6 Conclusions

- The power of the Myhill–Nerode–type formalism extends beyond the finite-state (i.e. related to finite automata) properties. Nice, isn’t it?

- Still, one would like to see an explicit (perhaps logic-based) framework for XP algorithms, analogously to the MSOL framework with FPT algorithms, cf. [Courcelle] et al.
  
  - Our presented unified approach shows this should be possible.

  - And very recently, [Král’, Obdržálek, and PH, 2010] have succeeded in finding such a framework.

- **BTW** (totally unrelated...)

  Have you already heard that the crossing number of almost planar graphs is NP-complete? [Cabello and Mohar, 2010]

THANK YOU FOR YOUR ATTENTION