Graph decompositions, Parse trees, and MSO properties

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 Motivation, and a short survey

Algorithmics. Many hard graph problems become easy on *trees*. . .

Any natural problem which is NP-hard on trees?

1 Motivation, and a short survey

Algorithmics. Many hard graph problems become easy on *trees*. . .

Any natural problem which is NP-hard on trees? Bandwidth.

- More generaly, many problems are easy on (partial) k-*trees*, i.e. on the graphs of bounded *tree-width* [Arnborg et al, 80's].

1 Motivation, and a short survey

Algorithmics. Many hard graph problems become easy on *trees*...

Any natural problem which is NP-hard on trees? Bandwidth.

- More generaly, many problems are easy on (partial) k-*trees*, i.e. on the graphs of bounded *tree-width* [Arnborg et al, 80's].
- In what other ways "*similarlity to trees*" can be defined?

1 Motivation, and a short survey

Algorithmics. Many hard graph problems become easy on *trees*. . .

Any natural problem which is NP-hard on trees? Bandwidth.

- More generaly, many problems are easy on (partial) k-*trees*, i.e. on the graphs of bounded *tree-width* [Arnborg et al, 80's].
- In what other ways "*similarlity to trees*" can be defined? See later. . .

Theory. [Robertson and Seymour, Graph minors 80's]

Tree-decompositions present a core tool in this deep theory.

1 Motivation, and a short survey

Algorithmics. Many hard graph problems become easy on *trees*...

Any natural problem which is NP-hard on trees? Bandwidth.

- More generaly, many problems are easy on (partial) k-*trees*, i.e. on the graphs of bounded *tree-width* [Arnborg et al, 80's].
- In what other ways "*similarlity to trees*" can be defined? See later...

Theory. [Robertson and Seymour, Graph minors 80's]

Tree-decompositions present a core tool in this deep theory.

- This theory started wide interest in tree-width in the CS community...

What is Tree-Width?

- The *tree-width* of a graph G equals the smallest possible clique size minus one of a chordal supergraph of G.

What is Tree-Width?

- The *tree-width* of a graph G equals the smallest possible clique size minus one of a chordal supergraph of G.
- A really useful definition, isn't it?

What is Tree-Width?

- The *tree-width* of a graph G equals the smallest possible clique size minus one of a chordal supergraph of G.
- A really useful definition, isn't it?
- OK, let us try once more... [Robertson and Seymour, 80's]

Definition. A *tree-decomposition* of a graph G is a tree with

– "bags" (subsets) of vertices of G assigned to the tree nodes,

– each edge of G belonging to some bag, and

What is Tree-Width?

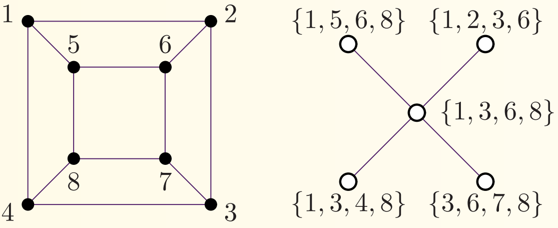
- The *tree-width* of a graph G equals the smallest possible clique size minus one of a chordal supergraph of G.
- A really useful definition, isn't it?
- OK, let us try once more... [Robertson and Seymour, 80's]

Definition. A *tree-decomposition* of a graph G is a tree with

– "bags" (subsets) of vertices of G assigned to the tree nodes,

– each edge of G belonging to some bag, and

– the bags containing some vertex must form a subtree (interpolation).



What is Tree-Width?

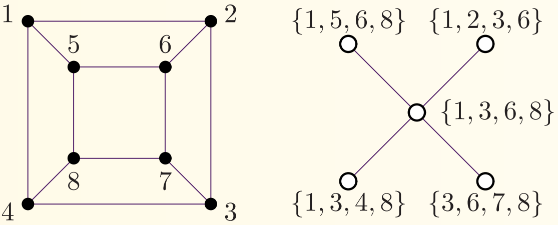
- The *tree-width* of a graph G equals the smallest possible clique size minus one of a chordal supergraph of G.
- A really useful definition, isn't it?
- OK, let us try once more... [Robertson and Seymour, 80's]

Definition. A *tree-decomposition* of a graph G is a tree with

– "bags" (subsets) of vertices of G assigned to the tree nodes,

– each edge of G belonging to some bag, and

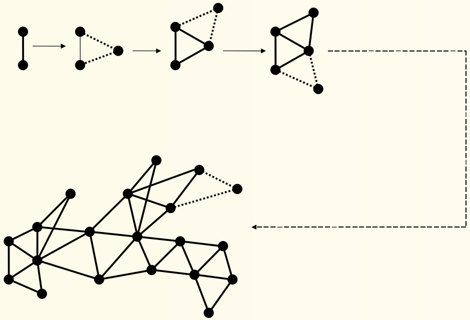
– the bags containing some vertex must form a subtree (interpolation).



Tree-width $=\min_{\text{decomps. of } G} \max\left\{|B|-1 : B \text{ bag in decomp.}\right\}$

Alternative approach

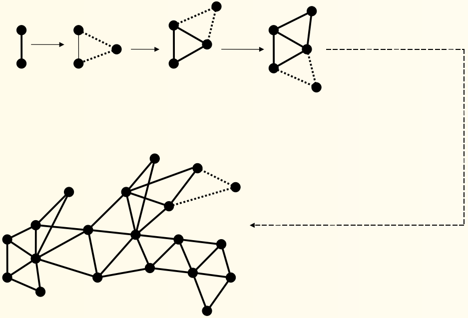
- Independently of R+S, tree-like decomposition have been approached via *k-trees*, see e.g. a 2-tree:



[Beineke & Pippert, 68 – 69], [Rose 74], [Arnborg & Proskurowski, 86].

Alternative approach

- Independently of R+S, tree-like decomposition have been approached via *k-trees*, see e.g. a 2-tree:

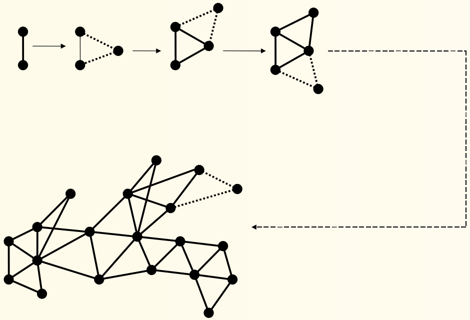


[Beineke & Pippert, 68 – 69], [Rose 74], [Arnborg & Proskurowski, 86].

- A graph G has tree-width $\leq k$ iff G is a partial (subgraph of a) *k-tree*.

Alternative approach

- Independently of R+S, tree-like decomposition have been approached via *k-trees*, see e.g. a 2-tree:



  [Beineke & Pippert, 68 – 69], [Rose 74], [Arnborg & Proskurowski, 86].

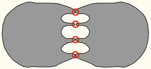
- A graph G has tree-width $\leq k$ iff G is a partial (subgraph of a) *k-tree*.

- Furthermore, k-trees easily relate tree-width to simplicial vertices and elimination orderings of chordal graphs.

Related notion: Branch-Width

- We want to measure *connectivity* of a graph G via edges $X \subseteq E(G)$:

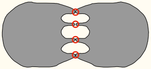
  $\boldsymbol{\lambda_G(X)} = \#$ vertices shared between X and $E(G) - X$.



Related notion: Branch-Width

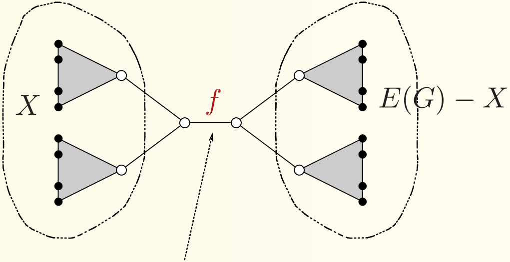
- We want to measure *connectivity* of a graph G via edges $X \subseteq E(G)$:

  $\boldsymbol{\lambda_G(X)} \;=\; \#$ vertices shared between X and $E(G) - X$.



- [Robertson and Seymour] – in analogy to tree-width. . .

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

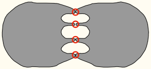


width$(e) \;=\; \lambda_G(X)$ where X is displayed by f in the tree.

Related notion: Branch-Width

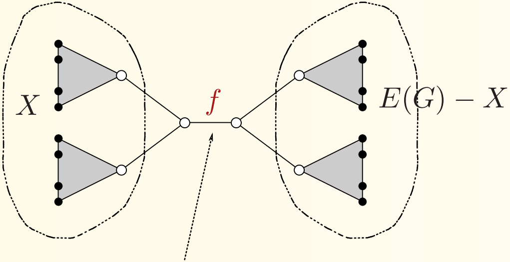
- We want to measure *connectivity* of a graph G via edges $X \subseteq E(G)$:

  $\lambda_G(X) \;=\; \#$ vertices shared between X and $E(G) - X$.



- [Robertson and Seymour] – in analogy to tree-width. . .

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



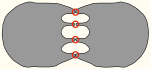
width$(e) \;=\; \lambda_G(X)$ where X is displayed by f in the tree.

Branch-width $= \; \mathbf{min}_{\text{branch-decs. of } G} \; \mathbf{max} \left\{ \text{width}(f) : f \text{ tree edge} \right\}$

Related notion: Branch-Width

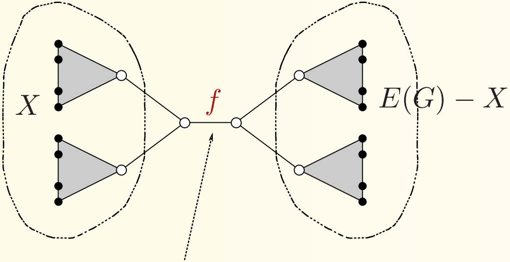
- We want to measure *connectivity* of a graph G via edges $X \subseteq E(G)$:

  $\boldsymbol{\lambda_G(X)} = \#$ vertices shared between X and $E(G) - X$.



- [Robertson and Seymour] – in analogy to tree-width...

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



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

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

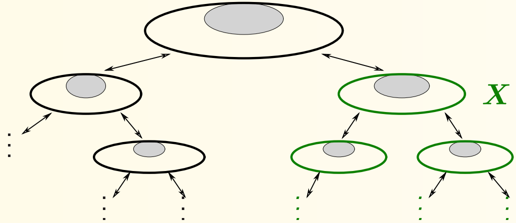
- Branch-width is within a constant factor of tree-width.

Fast Dynamic Algorithms

Example. Finding the largest *independent set* in a graph of tree-width at most k, assuming a rooted tree-decomposition is given, in time $\boldsymbol{O(2^k \cdot n)}$.

Fast Dynamic Algorithms

Example. Finding the largest *independent set* in a graph of tree-width at most k, assuming a rooted tree-decomposition is given, in time $O(2^k \cdot n)$.



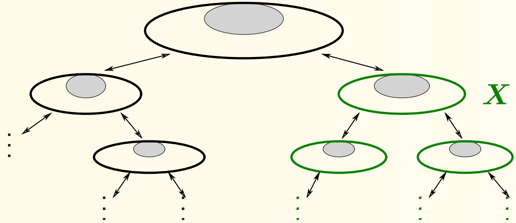
- In a bottom-up tree processing we collect this information:

$\mathcal{I}_X :$ $Y \subseteq$ decomposition bag $X \;\longrightarrow$

$$\mathbf{max}\ \big|\ \text{independent set } S \text{ ``}\textit{below}\text{'' } X \text{ s.t. } S \cap X = Y\ \big|$$

Fast Dynamic Algorithms

Example. Finding the largest *independent set* in a graph of tree-width at most k, assuming a rooted tree-decomposition is given, in time $O(2^k \cdot n)$.



- In a bottom-up tree processing we collect this information:

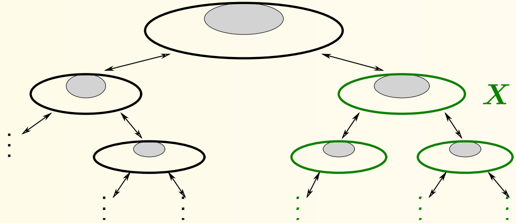
  $\mathcal{I}_X$ **:** $Y \subseteq$ decomposition bag $X \;\longrightarrow$
  $$\mathbf{max}\ \big|\ \text{independent set } S \text{ "}\textit{below}\text{" } X \text{ s.t. } S \cap X = Y\ \big|$$

- Computable by brute force at the leaves,

  and then straightforwardly combined together at internal nodes. . .

Fast Dynamic Algorithms

Example. Finding the largest *independent set* in a graph of tree-width at most k, assuming a rooted tree-decomposition is given, in time $\boldsymbol{O(2^k \cdot n)}$.



- In a bottom-up tree processing we collect this information:

  $\mathcal{I}_X$ **:** $Y \subseteq$ decomposition bag $X \;\longrightarrow$

  $$\mathbf{max}\ \big|\ \text{independent set } S \text{ “}\textit{below}\text{” } X \text{ s.t. } S \cap X = Y\ \big|$$

- Computable by brute force at the leaves,

  and then straightforwardly combined together at internal nodes...

- Total computing time: $O(2^k)$ times $O(n)$ nodes of the decomposition.

- Analogous dynamic (FPT) algorithms exist for, say, the dominating set, vertex cover, chromatic number, Hamiltonian cycle, etc. . .

Furthermore:

- Analogous dynamic (FPT) algorithms exist for, say, the dominating set, vertex cover, chromatic number, Hamiltonian cycle, etc. . .

Furthermore:

Theorem. [Courcelle 88], [Arnborg, Lagergren, and Seese, 88]

All graph properties expressible in *MSO logic* (MS_2 – vertices and edges) on the graphs of bounded tree-width can be solved in FPT time $\boldsymbol{O(f(k) \cdot n)}$.

2 Parse Trees, a not-much-known tool

Assume a graph G with a given rooted tree-decomposition of with k.

- A typical idea for a *dynamic algorithm* on a tree-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 Parse Trees, a not-much-known tool

Assume a graph G with a given rooted tree-decomposition of with k.

- A typical idea for a *dynamic algorithm* on a tree-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 Parse Trees, a not-much-known tool

Assume a graph G with a given rooted tree-decomposition of with k.

- A typical idea for a *dynamic algorithm* on a tree-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 Parse Trees, a not-much-known tool

Assume a graph G with a given rooted tree-decomposition of with k.

- A typical idea for a *dynamic algorithm* on a tree-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 right congruence appeared in the works [Abrahamson and Fellows, 93], [Downey and Fellows, 99], and [PH, 03].

Canonical Equivalence on graphs

How does a right congruence extend

from formal words with the concatention operation

to, say, graphs with a kind of “join” operation?

Canonical Equivalence on graphs

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

Canonical Equivalence on graphs

How does a right congruence extend
from formal words with the concatention operation
to, say, graphs with a kind of "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 $H \in \mathcal{U}_k$,

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

Canonical Equivalence on graphs

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

The task now is to make our join operation to "play with"

the decomposition we have of our graph...

Parse Trees of decompositions

The task now is to make our join operation to "play with"
the decomposition we have of our graph...

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

boundary size k	$\leftrightarrow$	restricted bag-size / width in decomposition
join operator $\oplus$	$\leftrightarrow$	the way pieces of G "stick together" in decomp.

Parse Trees of decompositions

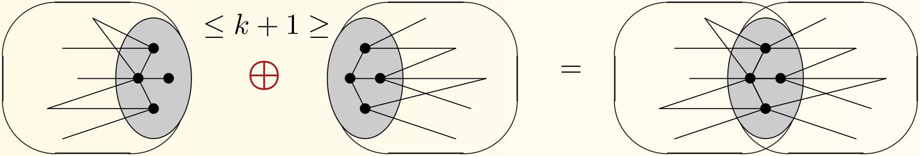
The task now is to make our join operation to "play with"
the decomposition we have of our graph...

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

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

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

- E.g. for a tree-decomposition of width k:



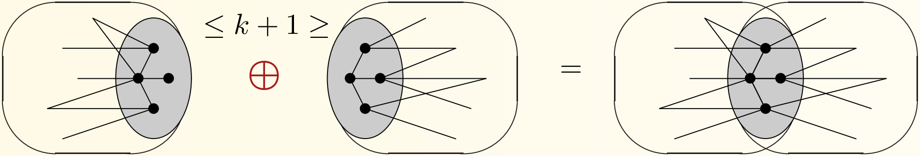
Parse Trees of decompositions

The task now is to make our join operation to "play with"
the decomposition we have of our graph...

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

boundary size k	$\leftrightarrow$	restricted bag-size / width in decomposition
join operator $\oplus$	$\leftrightarrow$	the way pieces of G "stick together" in decomp.

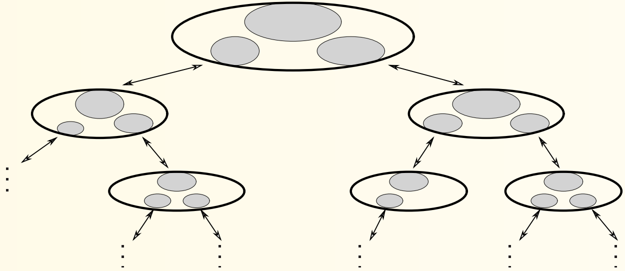
- E.g. for a tree-decomposition of width k:



(Similarly for a branch-decomposition, but without sharing bd. edges.)

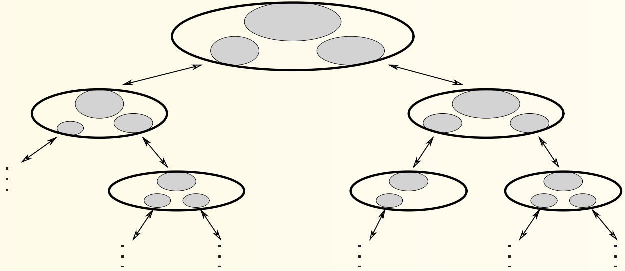
- A *boundaried parse tree* is then obtained as a

  "translation" of the decomposition into the above meaning of a *boundary* and a *join operation* (actually extended to a composition operator).



- A *boundaried parse tree* is then obtained as a

  "translation" of the decomposition into the above meaning of a *boundary* and a *join operation* (actually extended to a composition operator).



- Now, mod. some technical assumptions on parse trees and $\oplus$, we can get:

Theorem. (Analogy of [Myhill–Nerode])

$\mathcal{P}$ is accepted by a finite tree automaton on parse trees of boundary size $\leq k$

if and only if $\approx_{\mathcal{P},k}$ has finitely many classes on $\mathcal{U}_k$.

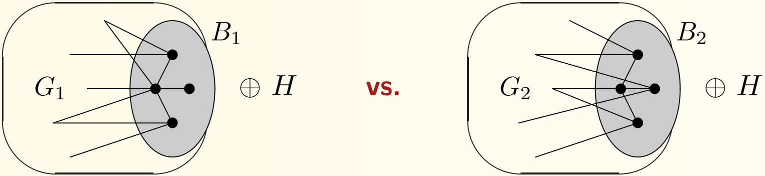
Example. $\mathcal{P} = \mathcal{C}_3$: *3-colourability* of graphs of tree-width $\leq k$.

Example. $\mathcal{P} = \mathcal{C}_3$: *3-colourability* of graphs of tree-width $\leq k$.

- For G_i with boundary $B_i \subseteq V(G_i)$ s.t. $|B_i| \leq k+1$, $i = 1, 2$, we have

  $(G_1, B_1) \approx_{\mathcal{C}_3,k} (G_2, B_2)$ if and only if

  $$\left\{ \chi \restriction B_1 : \chi \text{ prop. 3-col. } G_1 \right\} = \left\{ \chi \restriction B_2 : \chi \text{ prop. 3-col. } G_2 \right\}.$$

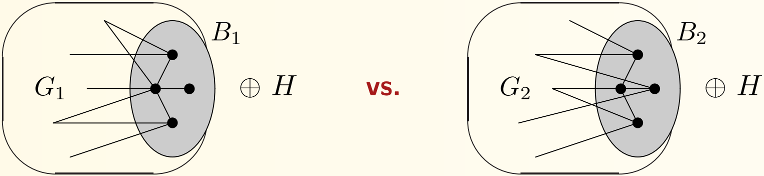


Example. $\mathcal{P} = \mathcal{C}_3$: *3-colourability* of graphs of tree-width $\leq k$.

- For G_i with boundary $B_i \subseteq V(G_i)$ s.t. $|B_i| \leq k+1$, $i = 1, 2$, we have

  $(G_1, B_1) \approx_{\mathcal{C}_3,k} (G_2, B_2)$ if and only if

  $\{\chi \restriction B_1 : \chi$ prop. 3-col. $G_1\} = \{\chi \restriction B_2 : \chi$ prop. 3-col. $G_2\}$.



- Then $\approx_{\mathcal{C}_3,k}$ has finitely many classes, depending only on k

  – information "of size $O(3^k)$".

  That easily results in an $O(3^k n)$ FPT algorithm for 3-colourability!

Dynamic Algorithms revisited

- How to capture non-decision problems in the previous framework?

  – allow *free variables* in the property $\mathcal{Q}(X)$!

  E.g. $\mathcal{Q}(X) \equiv$ *independent*(X), *dominating*(X), or *matching*(X).

Dynamic Algorithms revisited

- How to capture non-decision problems in the previous framework?

  – allow *free variables* in the property $\mathcal{Q}(X)$!

  E.g. $\mathcal{Q}(X) \equiv$ *independent*(X), *dominating*(X), or *matching*(X).

Definition. *Extended canonical equivalence* $\boldsymbol{\approx_{\mathcal{Q}(X),k}}$

– like $\approx_{\mathcal{P},k}$ on the univ. $\mathcal{U}_k[X]$ of graphs *equipped* with interpretation of X.

Dynamic Algorithms revisited

- How to capture non-decision problems in the previous framework?
  – allow *free variables* in the property $\mathcal{Q}(X)$!

  E.g. $\mathcal{Q}(X) \equiv$ *independent*(X), *dominating*(X), or *matching*(X).

Definition. *Extended canonical equivalence* $\approx_{\mathcal{Q}(X),k}$

– like $\approx_{\mathcal{P},k}$ on the univ. $\mathcal{U}_k[X]$ of graphs *equipped* with interpretation of X.

LinEMSO properties [Arnborg et al, 88], [Courcelle et al, 00].

– allowing MSO plus optimization and / or enumeration over *linear evaluational terms* in the free variables.

E.g. $\max|X|$: *independent*(X), or $\#X$: *matching*(X).

Dynamic Algorithms revisited

- How to capture non-decision problems in the previous framework?
  – allow *free variables* in the property $\mathcal{Q}(X)$!

  E.g. $\mathcal{Q}(X) \equiv$ *independent*(X), *dominating*(X), or *matching*(X).

Definition. *Extended canonical equivalence* $\approx_{\mathcal{Q}(X),k}$

– like $\approx_{\mathcal{P},k}$ on the univ. $\mathcal{U}_k[X]$ of graphs *equipped* with interpretation of X.

LinEMSO properties [Arnborg et al, 88], [Courcelle et al, 00].

– allowing MSO plus optimization and / or enumeration over *linear evaluational terms* in the free variables.

E.g. $\max |X|$: *independent*(X), or $\#X$: *matching*(X).

- Fitting into the *parse tree framework*:
  – In the dynamic programming paradigm, remember optimal representatives and / or partial enum. results for each class of the extended canonical equivalence.

Corollary. Besides, we get a straightforward inductive proof that:

All MSO formulas ϕ (even with *free variables*) generate finitely many classes of the ext. canonical equivalence $\approx_{\phi,k}$.

[Abrahamson and Fellows, 93], and [PH, 03].

– Clear for *atomic* predicates like $x \in X$ or *edge*(x, y) (cf. boundary k !).

Corollary. Besides, we get a straightforward inductive proof that:

All MSO formulas ϕ (even with *free variables*) generate finitely many classes of the ext. canonical equivalence $\approx_{\phi,k}$.

[Abrahamson and Fellows, 93], and [PH, 03].

- Clear for *atomic* predicates like $x \in X$ or *edge*(x, y) (cf. boundary k !).
- Then process $\neg\phi$, $\phi \vee \psi$ (easy),

Corollary. Besides, we get a straightforward inductive proof that:

All MSO formulas ϕ (even with *free variables*) generate finitely many classes of the ext. canonical equivalence $\approx_{\boldsymbol{\phi},\boldsymbol{k}}$.

[Abrahamson and Fellows, 93], and [PH, 03].

- Clear for *atomic* predicates like $x \in X$ or *edge*(x, y) (cf. boundary k !).
- Then process $\neg\phi$, $\phi \vee \psi$ (easy), or $\exists x\, \phi(x)$, $\exists X\, \phi(X)$ (quite hard, need an exponential jump in the number of classes with each quantification!).

3 Rank-Width and Parse trees

Some other views of being "similar to trees"...

- *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.

3 Rank-Width and Parse trees

Some other views of being "similar to trees"...

- *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.

- Clique-width shares some nice properties with tree-width, e.g.

  Theorem. [Courcelle, Makowsky, and Rotics 00]

  All graph properties expressible in *MSO logic* (MS_1 – only vertices!!!) on the graphs of bounded clique-width can be solved in time $\boldsymbol{O(f(k) \cdot n)}$.

3 Rank-Width and Parse trees

Some other views of being "similar to trees"...

- *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.
- Clique-width shares some nice properties with tree-width, e.g.

  Theorem. [Courcelle, Makowsky, and Rotics 00]

  All graph properties expressible in *MSO logic* (MS_1 – only vertices!!!) on the graphs of bounded clique-width can be solved in time $\boldsymbol{O(f(k) \cdot n)}$.
- On the other hand, clique-width has some drawbacks,

  like we do not know how to test clique-width k if $k \geq 3$.

Rank-Decompositions

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

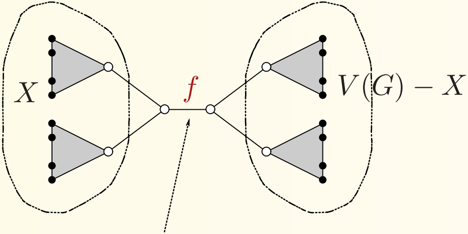
$$\varrho_G(X) = \text{rank of} \quad \begin{array}{c} \\ X \end{array} \overset{V(G)-X}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}} \text{ modulo } 2$$

Rank-Decompositions

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

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

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



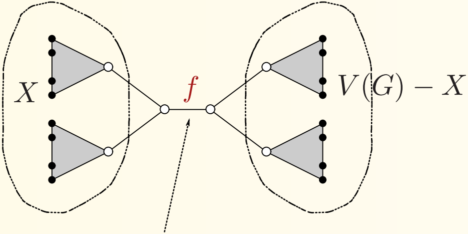
width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

Rank-Decompositions

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

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

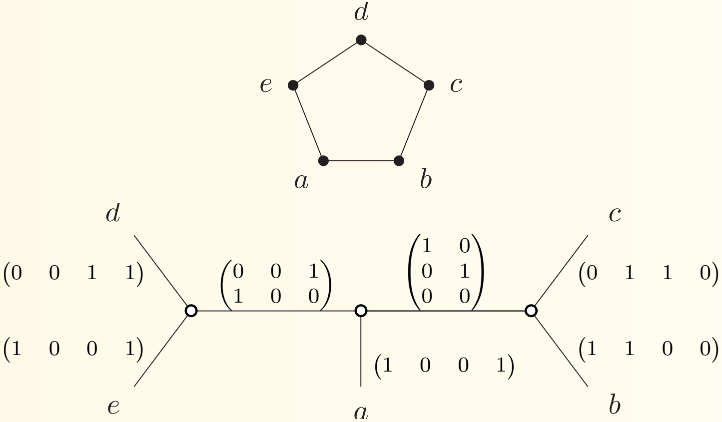
Definition. Decompose $V(G)$ one-to-one into the leaves of a subcubic tree. Then:



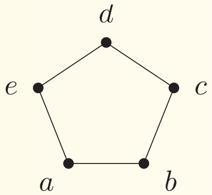
width$(e) = \varrho_G(X)$ where X is displayed by f in the tree.

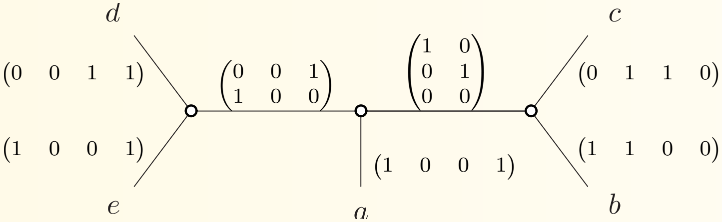
Rank-width $= \min_{\text{rank-decs. of } G} \max \{\text{width}(f) : f \text{ tree edge}\}$

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



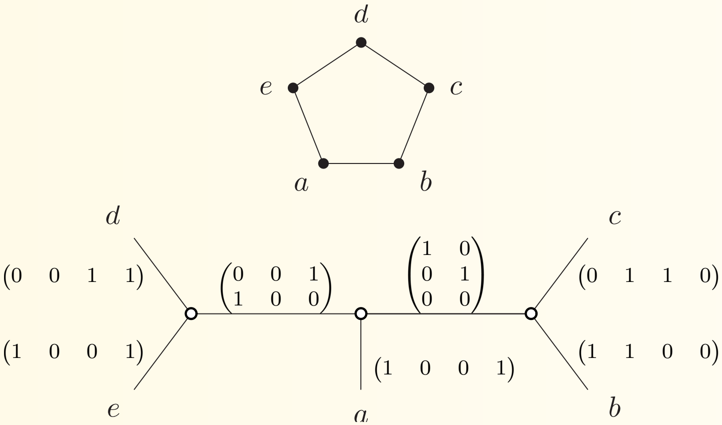
- An example: cycle C_5 and its *rank-decomposition* of width 2:





- Rank-width t is related to clique-width k: $\boldsymbol{k \leq t \leq 2^{k+1} - 1}$

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



- Rank-width t is related to clique-width k: $\boldsymbol{k \leq t \leq 2^{k+1} - 1}$
- [Oum and PH, 07] There is an FPT algorithm for computing an optimal rank-decomposition of a graph in time $O(f(t) \cdot n^3)$.

Boundary and Join for rank-decompositions

Unlike 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" has just an impl. matrix rank!

Boundary and Join for rank-decompositions

Unlike 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" has just an impl. matrix rank!

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

Boundary and Join for rank-decompositions

Unlike 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" has just an impl. 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 $GF(2)^t$ s.t.

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

Boundary and Join for rank-decompositions

Unlike 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" has just an impl. 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 $GF(2)^t$ s.t.

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

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

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

$\to$ rank-width *parse tree* [Ganian and PH, 08].

Boundary and Join for rank-decompositions

Unlike 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" has just an impl. 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 $GF(2)^t$ s.t.

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

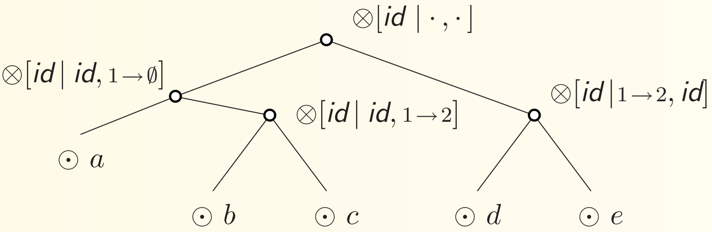
- Join $\;\to\;$ *composition* operator with relabelings f_1, f_2:

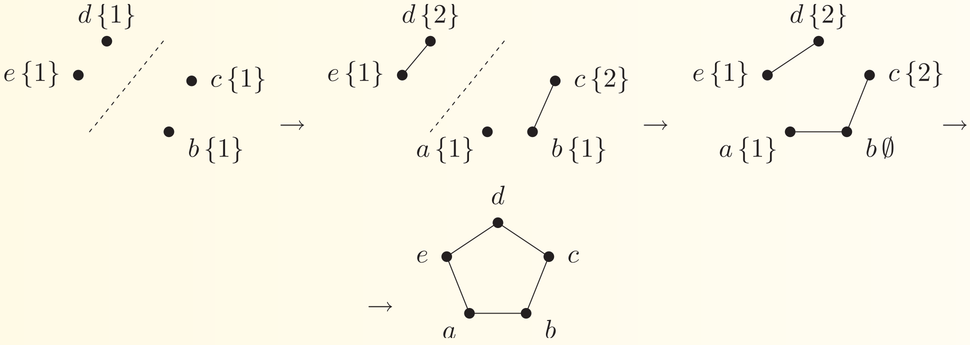
$$(G_1, lab^1) \;\otimes[\mathbf{g} \mid f_1, f_2]\; (G_2, lab^2) \;=\; (H, lab)$$

  $\to$ rank-width *parse tree* [Ganian and PH, 08].

- 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):





Algorithms on bounded Rank-Width

Bare rank-decomposition — not enough information for dynamic algorithms. . .

- Stronger *parse trees* give needed extra information for algorithms!

Algorithms on bounded Rank-Width

Bare rank-decomposition — not enough information for dynamic algorithms...

- Stronger *parse trees* give needed extra information for algorithms!
  - With "boundary" and "join" at hand, we get the associated canonical equivalence classes.
  - Again, the whole *LinEMSO* framework fits here nicely...

Algorithms on bounded Rank-Width

Bare rank-decomposition — not enough information for dynamic algorithms. . .

- Stronger *parse trees* give needed extra information for algorithms!
  - With "boundary" and "join" at hand, we get the associated canonical equivalence classes.
  - Again, the whole *LinEMSO* framework fits here nicely. . .

- **Example:** the 3-colourability problem.

  For G_i with *t-labeling* ($\sim$boundary) $lab^i : V(G_i) \to \{1, \ldots, t\}$, $i = 1, 2$, we have

$$(G_1, lab^1) \approx_{\mathcal{C}_{3,t}} (G_2, lab^2) \text{ if}$$

$$\Big\{ \big(lab^1(\chi^{-1}(i)) : i = 1,2,3 \big) : \chi \text{ prop. 3-col. } G_1 \Big\} =$$
$$= \Big\{ \big(lab^2(\chi^{-1}(i)) : i = 1,2,3 \big) : \chi \text{ prop. 3-col. } G_2 \Big\}.$$

Algorithms on bounded Rank-Width

Bare rank-decomposition — not enough information for dynamic algorithms...

- Stronger *parse trees* give needed extra information for algorithms!
  - With "boundary" and "join" at hand, we get the associated canonical equivalence classes.
  - Again, the whole *LinEMSO* framework fits here nicely...

- **Example:** the 3-colourability problem.

  For G_i with *t-labeling* (∼boundary) $lab^i : V(G_i) \to \{1,\dots,t\}$, $i = 1,2$, we have

$$(G_1, lab^1) \approx_{\mathcal{C}_{3,t}} (G_2, lab^2) \text{ if}$$

$$\left\{\left(lab^1(\chi^{-1}(i)) : i = 1,2,3\right) : \chi \text{ prop. 3-col. } G_1\right\} = $$
$$= \left\{\left(lab^2(\chi^{-1}(i)) : i = 1,2,3\right) : \chi \text{ prop. 3-col. } G_2\right\}.$$

  This readily gives an FPT $O(f(t) \cdot n)$ algorithm.

4 Final remarks

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

4 Final remarks

- Parse trees appear a useful tool in 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.
  - not being considered in depth so far...

4 Final remarks

- Parse trees appear a useful tool in 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.
  - not being considered in depth so far...

- Ongoing work
  - indentify a (useful) fragment of MSO with “small” number of classes (in general, each quantifier alternation makes an exponential jump [Frick and Grohe, 04]),

4 Final remarks

- Parse trees appear a useful tool in 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.
  - not being considered in depth so far...
- Ongoing work
  - indentify a (useful) fragment of MSO with "small" number of classes (in general, each quantifier alternation makes an exponential jump [Frick and Grohe, 04]),
  - this appears useful especially in connection with rank-width...

4 Final remarks

- Parse trees appear a useful tool in 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.
  - not being considered in depth so far...
- Ongoing work
  - indentify a (useful) fragment of MSO with "small" number of classes (in general, each quantifier alternation makes an exponential jump [Frick and Grohe, 04]),
  - this appears useful especially in connection with rank-width...

THANK YOU FOR YOUR ATTENTION