On an odd case of an XP algorithm for graphs of bounded clique-width

Petr Hliněný

Faculty of Informatics
Masaryk University, Brno, CZ

Based on joint work with
R. Ganian and J. Obdržálek,
orig. presented at STACS 2011.

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused. . .

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused...

"Store and reuse" can be handled in various ways

– we are dealing with the static approach (*tabulation*).

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused...

"Store and reuse" can be handled in various ways

– we are dealing with the static approach (*tabulation*).

Using tables

Determine, which subproblems are *significant!*

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused...

"Store and reuse" can be handled in various ways

– we are dealing with the static approach (*tabulation*).

Using tables

Determine, which subproblems are *significant!*

– in the sense of giving a partition over all subproblems into "same-behavior" classes (guar. to give the same outcome).

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused...

"Store and reuse" can be handled in various ways

– we are dealing with the static approach (*tabulation*).

Using tables

Determine, which subproblems are *significant!*

– in the sense of giving a partition over all subproblems into "same-behavior" classes (guar. to give the same outcome).

The big hidden problem

Assuming such reasonably described *canonical classes*;

(say, of annotated subproblems)

1 Introduction: What is it about?

Dynamic programming

Recursive computation of the whole solution from subproblems, such that subsolutions are stored once and then reused...

"Store and reuse" can be handled in various ways

– we are dealing with the static approach (*tabulation*).

Using tables

Determine, which subproblems are *significant!*

– in the sense of giving a partition over all subproblems into "same-behavior" classes (guar. to give the same outcome).

The big hidden problem

Assuming such reasonably described *canonical classes*;

(say, of annotated subproblems)

can one always process those throughout the recursion?

More Formally...

Inspiration – classical Myhill–Nerode

Finite automaton states $\iff$
right congruence classes on words (of a regular language).

More Formally. . .

Inspiration – classical Myhill–Nerode

Finite automaton states $\iff$

right congruence classes on words (of a regular language).

Abstract setting

- Recursive decomposition of the input, along a suit. *join* $\otimes$ oper.,

More Formally. . .

Inspiration – classical Myhill–Nerode

Finite automaton states $\iff$
right congruence classes on words (of a regular language).

Abstract setting

- Recursive decomposition of the input, along a suit. *join* $\otimes$ oper.,
- hence a universe of annot. subproblems $\mathcal{U} = \{(G, \varphi), \dots\}$,
(say, ϕ represents a solution fragment)

More Formally. . .

Inspiration – classical Myhill–Nerode

Finite automaton states $\iff$

right congruence classes on words (of a regular language).

Abstract setting

- Recursive decomposition of the input, along a suit. *join* $\otimes$ oper.,
- hence a universe of annot. subproblems $\mathcal{U} = \{(G, \varphi), \dots\}$,
  (say, ϕ represents a solution fragment)
- and a *property* $\mathcal{P}$ $\rightarrow$ feasibility of a (sub)solution.

More Formally. . .

Inspiration – classical Myhill–Nerode

Finite automaton states $\iff$

right congruence classes on words (of a regular language).

Abstract setting

- Recursive decomposition of the input, along a suit. *join* $\otimes$ oper.,
- hence a universe of annot. subproblems $\mathcal{U} = \{(G, \varphi), \dots\}$,

  (say, ϕ represents a solution fragment)
- and a *property* $\mathcal{P}$ $\rightarrow$ feasibility of a (sub)solution.

Canonical equivalence; metadefinition Abrahamson–Fellows

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

$(G_1, \varphi_1) \approx_{\mathcal{P}} (G_2, \varphi_2)$ if and only if, for all (H, φ),

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

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

$\rightarrow$ *canonical classes* $X \in \mathcal{X}$,

$\rightarrow$ these must refine the equiv. classes of $\approx_{\mathcal{P}}$, but may be finer.

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

$\rightarrow$ *canonical classes* $X \in \mathcal{X}$,

$\rightarrow$ these must refine the equiv. classes of $\approx_{\mathcal{P}}$, but may be finer.

Processing canonical classes

– By the definition, all we need to know about a solution of an annot. subproblem, is the class $X \in \mathcal{X}$ it belongs to!

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

$\rightarrow$ *canonical classes* $X \in \mathcal{X}$,

$\rightarrow$ these must refine the equiv. classes of $\approx_{\mathcal{P}}$, but may be finer.

Processing canonical classes

– By the definition, all we need to know about a solution of an annot. subproblem, is the class $X \in \mathcal{X}$ it belongs to!

– Hence "store and reuse" representat. solutions of each $X \in \mathcal{X}$.
$\rightarrow$ immediate tabulation... Easy as that?

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

$\rightarrow$ *canonical classes* $X \in \mathcal{X}$,

$\rightarrow$ these must refine the equiv. classes of $\approx_{\mathcal{P}}$, but may be finer.

Processing canonical classes

- By the definition, all we need to know about a solution of an annot. subproblem, is the class $X \in \mathcal{X}$ it belongs to!
- Hence "store and reuse" representat. solutions of each $X \in \mathcal{X}$.
  $\rightarrow$ immediate tabulation... Easy as that?

Consistency with $\otimes$? – stronger than just "honoring $\approx_{\mathcal{P}}$"

Canonical Classes

Definition – cf. the canonical equivalence of $\mathcal{P}$

A partition $\mathcal{X}$ of the univ. $\mathcal{U}$ is *canonical* if all $X \in \mathcal{X}$ honor $\approx_{\mathcal{P}}$.

$\rightarrow$ *canonical classes* $X \in \mathcal{X}$,

$\rightarrow$ these must refine the equiv. classes of $\approx_{\mathcal{P}}$, but may be finer.

Processing canonical classes

- By the definition, all we need to know about a solution of an annot. subproblem, is the class $X \in \mathcal{X}$ it belongs to!
- Hence "store and reuse" representat. solutions of each $X \in \mathcal{X}$.
  $\rightarrow$ immediate tabulation... Easy as that?

Consistency with $\otimes$? – stronger than just "honoring $\approx_{\mathcal{P}}$"

A canonical partition $\mathcal{X}$ is *consistent with* $\otimes$ if

- for all $(G_1, \varphi_1) \in X_1$, $(G_2, \varphi_2) \in X_2$;
  the part of $(G_1, \varphi_1) \otimes (G_2, \varphi_2)$ depends only on X_1, X_2.

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which

  $\rightarrow$ can be minimized, giving straight the equiv. classes of $\approx_{\mathcal{P}}$,

  $\rightarrow$ automatically consistent with $\otimes$!

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which

  $\rightarrow$ can be minimized, giving straight the equiv. classes of $\approx_{\mathcal{P}}$,

  $\rightarrow$ automatically consistent with $\otimes$!

- All this "hidden" in automaton formalism and $\mathcal{O}$-notation, FPT.

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which
  $\rightarrow$ can be minimized, giving straight the equiv. classes of $\approx_{\mathcal{P}}$,
  $\rightarrow$ automatically consistent with $\otimes$!
- All this "hidden" in automaton formalism and $\mathcal{O}$-notation, FPT.

Infinite Index $\approx_{\mathcal{P}}$

- Infinite table size? No, just growing with the input size...
  Hopefully *polynomial table size* $\rightarrow$ hoping for an XP algorithm.

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which
  $\rightarrow$ can be minimized, giving straight the equiv. classes of $\approx_{\mathcal{P}}$,
  $\rightarrow$ automatically consistent with $\otimes$!
- All this "hidden" in automaton formalism and $\mathcal{O}$-notation, FPT.

Infinite Index $\approx_{\mathcal{P}}$

- Infinite table size? No, just growing with the input size...
  Hopefully *polynomial table size* $\rightarrow$ hoping for an XP algorithm.
- Table update (wrt. $\otimes$) has to be done in polytime...

Finite Index $\approx_{\mathcal{P}}$

- As in Myhill–Nerode, there is a finite automaton which
  $\rightarrow$ can be minimized, giving straight the equiv. classes of $\approx_{\mathcal{P}}$,
  $\rightarrow$ automatically consistent with $\otimes$!
- All this "hidden" in automaton formalism and $\mathcal{O}$-notation, FPT.

Infinite Index $\approx_{\mathcal{P}}$

- Infinite table size? No, just growing with the input size...
  Hopefully *polynomial table size* $\rightarrow$ hoping for an XP algorithm.
- Table update (wrt. $\otimes$) has to be done in polytime...
- But what if $\mathcal{X}$ is inconsistent with $\otimes$ (i.e., cannot do table update), and we have no "better" canonical partition?

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

> Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

– Contains directed Hamiltonian path. . . (= one leaf)

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

– Contains directed Hamiltonian path... (= one leaf)

Short overview of MinLOB (not so much related to our talk)

- Polynomial on DAGs,

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

– Contains directed Hamiltonian path. . . (= one leaf)

Short overview of MinLOB (not so much related to our talk)

- Polynomial on DAGs,
- but NP-hard already for DFVS number 1.

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

– Contains directed Hamiltonian path. . . (= one leaf)

Short overview of MinLOB (not so much related to our talk)

- Polynomial on DAGs,
- but NP-hard already for DFVS number 1.
- Seems to resist nearly all useful parametrizations (except by *clique-width / rank-width*).

2 The Odd Case: MinLOB

Minimum leaf outbranching in a digraph D:

Find an *outbranching* (directed spanning out-tree) in D, minimizing the number of leaves.

– Contains directed Hamiltonian path… ($=$ one leaf)

Short overview of MinLOB (not so much related to our talk)

- Polynomial on DAGs,
- but NP-hard already for DFVS number 1.
- Seems to resist nearly all useful parametrizations (except by *clique-width / rank-width*).
- In MSO_2, only one "$\exists F$" above MSO_1.
  Hence if an extension of Courcelle–Makowsky–Rotics is sought (MSO_1 on graphs of bounded clique-width), then MinLOB should be understood first…

Clique Decompositions and Width

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

Clique Decompositions and Width

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

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

Clique Decompositions and Width

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

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

Clique-width – Courcelle and Olariu

- Defined by operations on vertex–labelled $(1, 2, \ldots, k)$ graphs:

Clique Decompositions and Width

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

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

Clique-width – Courcelle and Olariu

- Defined by operations on vertex–labelled $(1, 2, \dots, k)$ graphs:
  - create a new vertex with label i,
  - take the disjoint union of two labeled graphs,
  - add all edges / arcs between vertices of label i and label j,
  - and relabel all vertices with label i to have label j.

Clique Decompositions and Width

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

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

Clique-width – Courcelle and Olariu

- Defined by operations on vertex–labelled $(1, 2, \ldots, k)$ graphs:
  - create a new vertex with label i,
  - take the disjoint union of two labeled graphs,
  - add all edges / arcs between vertices of label i and label j,
  - and relabel all vertices with label i to have label j.

$\rightarrow$ Giving a rec. *decomposition* (parse tree) for clique-width.

Clique Decompositions and Width

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

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

Clique-width – Courcelle and Olariu

- Defined by operations on vertex–labelled $(1, 2, \ldots, k)$ graphs:
  - create a new vertex with label i,
  - take the disjoint union of two labeled graphs,
  - add all edges / arcs between vertices of label i and label j,
  - and relabel all vertices with label i to have label j.

$\rightarrow$ Giving a rec. *decomposition* (parse tree) for clique-width.

Labelled join $\otimes$ for clique-width

- Working over vertex–labelled graphs, too. Non-symmetric!

Clique Decompositions and Width

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

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

Clique-width – Courcelle and Olariu

- Defined by operations on vertex–labelled $(1, 2, \ldots, k)$ graphs:
  - create a new vertex with label i,
  - take the disjoint union of two labeled graphs,
  - add all edges / arcs between vertices of label i and label j,
  - and relabel all vertices with label i to have label j.

$\rightarrow$ Giving a rec. *decomposition* (parse tree) for clique-width.

Labelled join $\otimes$ for clique-width

- Working over vertex–labelled graphs, too. Non-symmetric!
- An "across" edge / arc depends only on the label of its end(s).

MinLOB on Clique-width

Annotating the subproblems

What is a solution fragment?

MinLOB on Clique-width

Annotating the subproblems

What is a solution fragment?

– having an *outforest*; i.e., a collection of many *out-trees* in G,

MinLOB on Clique-width

Annotating the subproblems

What is a solution fragment?

- having an *outforest*; i.e., a collection of many *out-trees* in G,
- "capture" its roots, and the *active* leaves—not to stay leaves (while the non-active ones contribute to the solution size);

MinLOB on Clique-width

Annotating the subproblems

What is a solution fragment?

– having an *outforest*; i.e., a collection of many *out-trees* in G,

– "capture" its roots, and the *active* leaves—not to stay leaves (while the non-active ones contribute to the solution size);

* actually, any vertex may be active, and even multiple times.

MinLOB on Clique-width

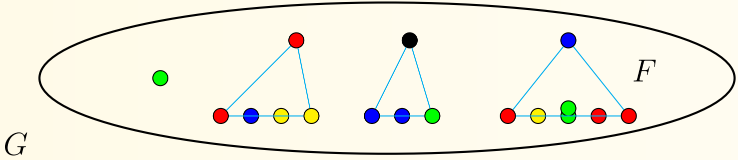
Annotating the subproblems

What is a solution fragment?

- having an *outforest*; i.e., a collection of many *out-trees* in G,
- "capture" its roots, and the *active* leaves—not to stay leaves (while the non-active ones contribute to the solution size);
  - * actually, any vertex may be active, and even multiple times.

Strong signature of an out-forest $F \subseteq G$ $\equiv$ for each tree of F;

- the root label, and the labels of all active vertices (a multiset).



MinLOB on Clique-width

Annotating the subproblems

What is a solution fragment?

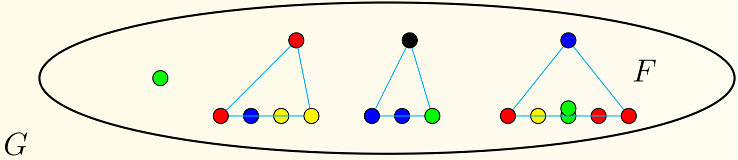
– having an *outforest*; i.e., a collection of many *out-trees* in G,

– "capture" its roots, and the *active* leaves—not to stay leaves (while the non-active ones contribute to the solution size);

* actually, any vertex may be active, and even multiple times.

Strong signature of an out-forest $F \subseteq G \equiv$ for each tree of F;

– the root label, and the labels of all active vertices (a multiset).



Fact. Strong signatures give canonical classes for MinLOB.

So, what is wrong here?

MinLOB on Clique-width

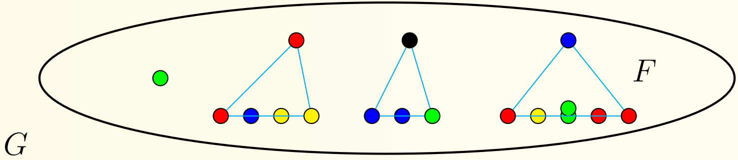
Annotating the subproblems

What is a solution fragment?

- having an *outforest*; i.e., a collection of many *out-trees* in G,
- "capture" its roots, and the *active* leaves—not to stay leaves (while the non-active ones contribute to the solution size);
  * actually, any vertex may be active, and even multiple times.

Strong signature of an out-forest $F \subseteq G$ $\equiv$ for each tree of F;

- the root label, and the labels of all active vertices (a multiset).

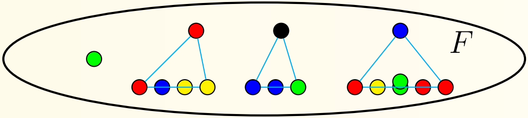


Fact. Strong signatures give canonical classes for MinLOB.

So, what is wrong here? The table is exponential!

3 Poly. Canonical Partition for MinLOB

Recall: Exp. *strong signature* of an out-forest $F \subseteq G$ = for each tree of F;
– the root label, and the labels of all active vertices (multiset).

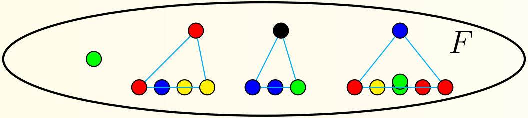


How to "reduce" this strong signature?

Signature (reduced) of an out-forest $F \subseteq G \equiv$ for each tree T of F;

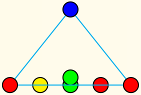
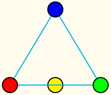
3 Poly. Canonical Partition for MinLOB

Recall: Exp. *strong signature* of an out-forest $F \subseteq G$ $=$ for each tree of F;
– the root label, and the labels of all active vertices (multiset).



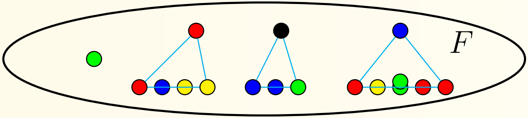
How to "reduce" this strong signature?

Signature (reduced) of an out-forest $F \subseteq G$ $\equiv$ for each tree T of F;
– the root label, and only the set of active v. labels occur. in T,

$\rightarrow$ i.e.,  $\longleftrightarrow$  of fixed size(!),

3 Poly. Canonical Partition for MinLOB

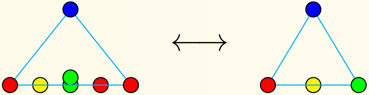
Recall: Exp. *strong signature* of an out-forest $F \subseteq G$ $=$ for each tree of F;
– the root label, and the labels of all active vertices (multiset).



How to "reduce" this strong signature?

Signature (reduced) of an out-forest $F \subseteq G$ $\equiv$ for each tree T of F;

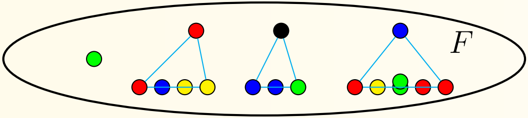
– the root label, and only the set of active v. labels occur. in T,

$\rightarrow$ i.e.,  of fixed size(!),

– and, separately, the multiset of active v. labels over whole F.

3 Poly. Canonical Partition for MinLOB

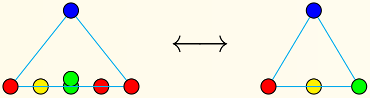
Recall: Exp. *strong signature* of an out-forest $F \subseteq G$ $=$ for each tree of F;
– the root label, and the labels of all active vertices (multiset).



How to "reduce" this strong signature?

Signature (reduced) of an out-forest $F \subseteq G$ $\equiv$ for each tree T of F;

– the root label, and only the set of active v. labels occur. in T,

$\rightarrow$ i.e.,  of fixed size(!),

– and, separately, the multiset of active v. labels over whole F.

Theorem. Even (reduced) signatures give canonical classes for MinLOB.

Proof

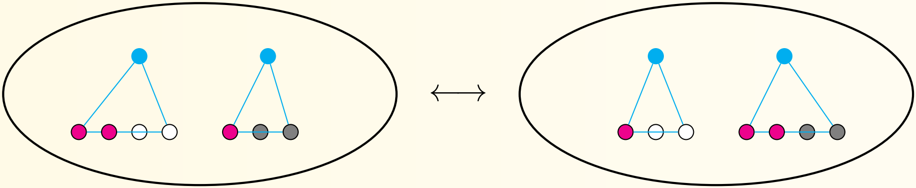
Theorem. Even (red.) signatures give canonical classes for MinLOB.

- Assume same signatures and different strong signatures.

Proof

Theorem. Even (red.) signatures give canonical classes for MinLOB.

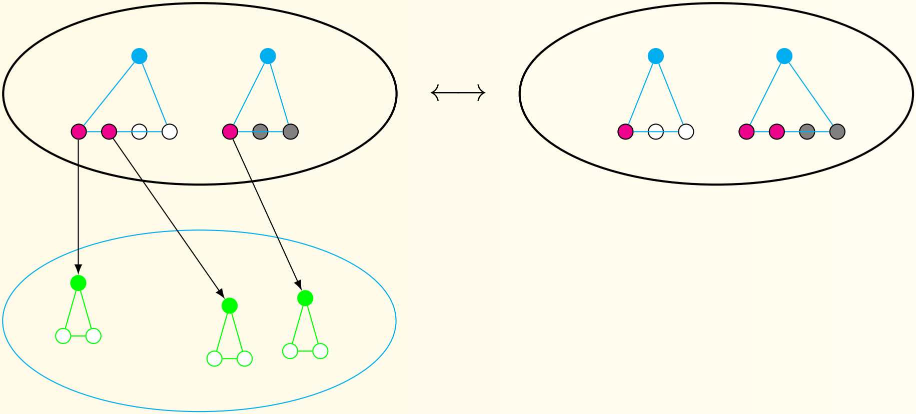
- Assume same signatures and different strong signatures.
- Minim. the diff. to just to ± 1 same-label occurrences in two trees:



Proof

Theorem. Even (red.) signatures give canonical classes for MinLOB.

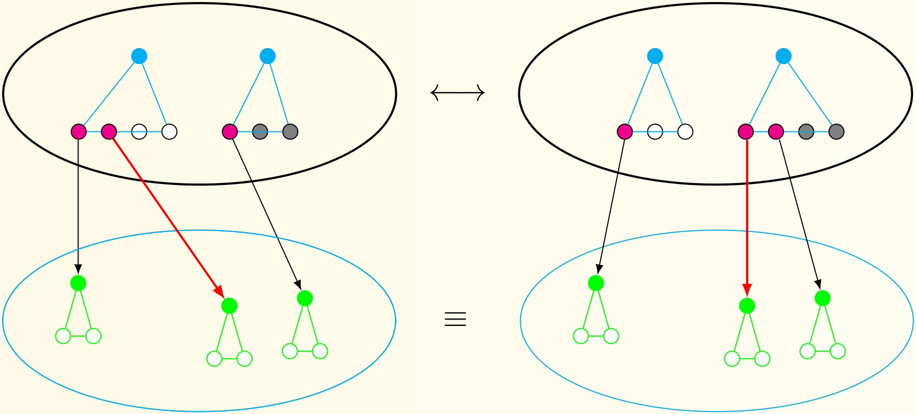
- Assume same signatures and different strong signatures.
- Minim. the diff. to just to ± 1 same-label occurrences in two trees:



Proof

Theorem. Even (red.) signatures give canonical classes for MinLOB.

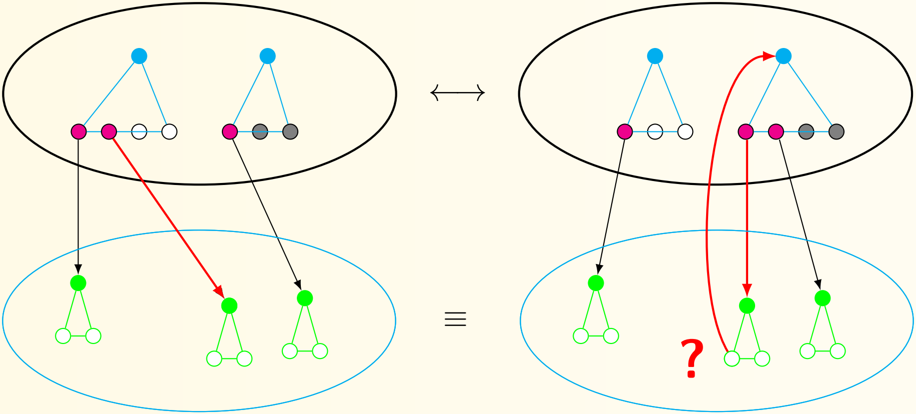
- Assume same signatures and different strong signatures.
- Minim. the diff. to just to ± 1 same-label occurrences in two trees:



Proof

Theorem. Even (red.) signatures give canonical classes for MinLOB.

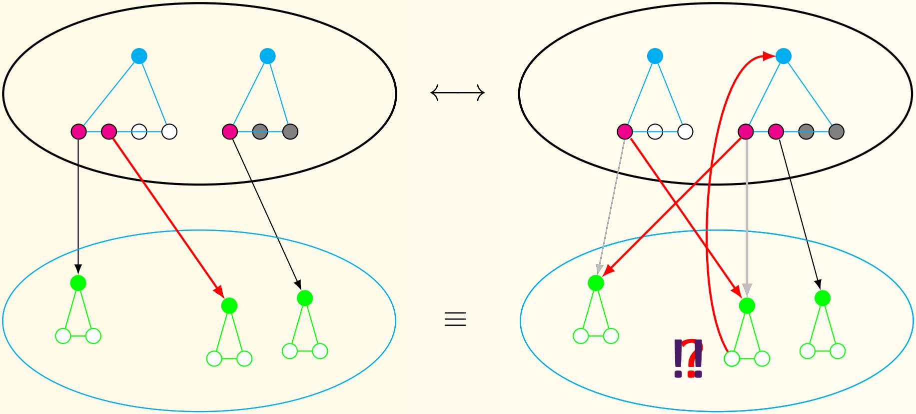
- Assume same signatures and different strong signatures.
- Minim. the diff. to just to ± 1 same-label occurrences in two trees:



Proof

Theorem. Even (red.) signatures give canonical classes for MinLOB.

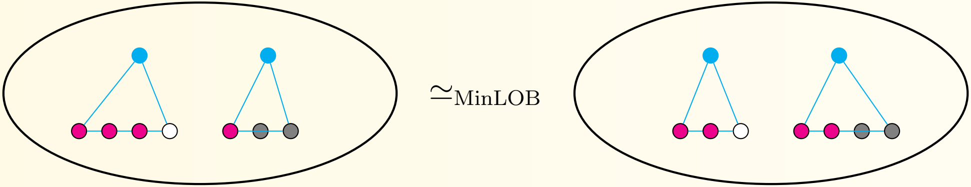
- Assume same signatures and different strong signatures.
- Minim. the diff. to just to ± 1 same-label occurrences in two trees:



Hence the upper two fragments are canonically equivalent, indeed. □

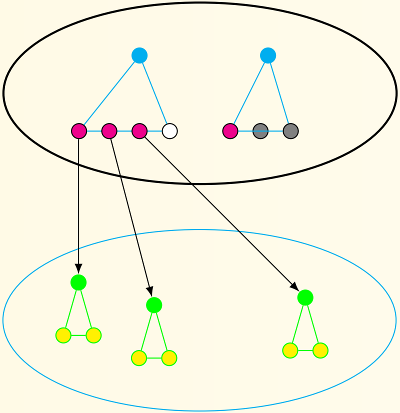
Now, What is Wrong?

Fact. Reduced signature is inconsistent with the labelled join $\otimes$!

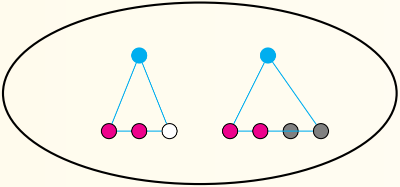


Now, What is Wrong?

Fact. Reduced signature is inconsistent with the labelled join $\otimes$!

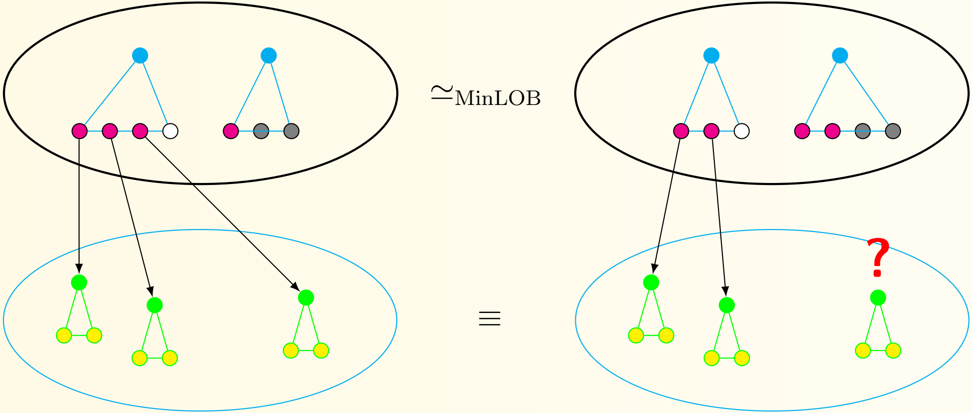


$\simeq_{\mathrm{MinLOB}}$



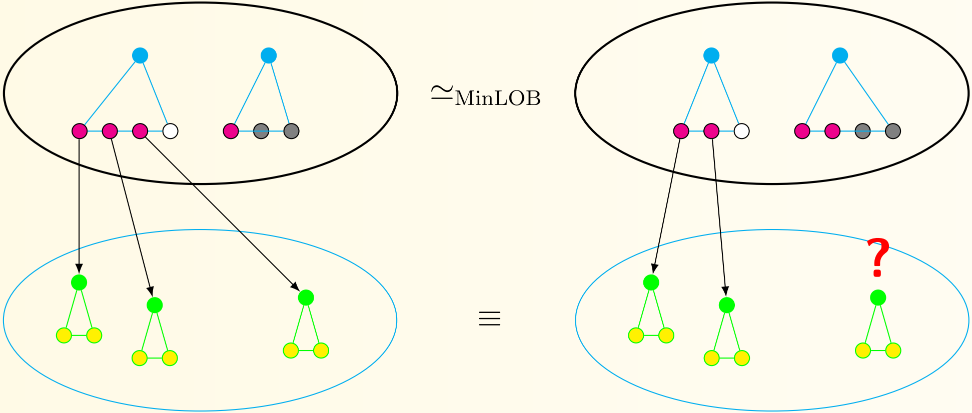
Now, What is Wrong?

Fact. Reduced signature is inconsistent with the labelled join $\otimes$!



Now, What is Wrong?

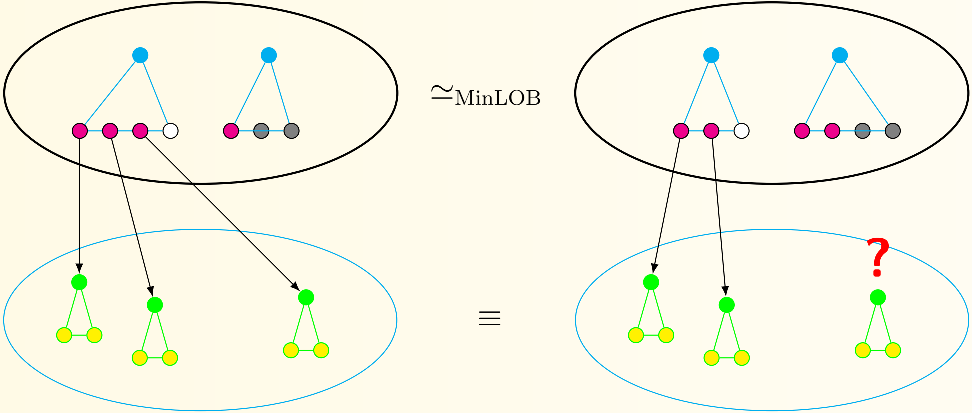
Fact. Reduced signature is inconsistent with the labelled join $\otimes$!



Any resolution? Simply ignore this inconsistency in dyn. algorithm...

Now, What is Wrong?

Fact. Reduced signature is inconsistent with the labelled join $\otimes$!



Any resolution? Simply ignore this inconsistency in dyn. algorithm...

- Pretend like if active labels do "not disappear" from particular tree, until all these labels are gone from the whole graph.

4 The XP Algorithm of Mystery

Tweaking the dynamic algorithm

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

4 The XP Algorithm of Mystery

Tweaking the dynamic algorithm

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

4 The XP Algorithm of Mystery

Tweaking the dynamic algorithm

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

Theorem. If a "singleton" weak signature is found on our decomposition, then the whole graph really contains an out-branching of the same number of leaves (constructively).

4 The XP Algorithm of Mystery

Tweaking the dynamic algorithm

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

Theorem. If a "singleton" weak signature is found on our decomposition, then the whole graph really contains an out-branching of the same number of leaves (constructively).

!!

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

4 The XP Algorithm of Mystery

Tweaking the dynamic algorithm

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

Theorem. If a "singleton" weak signature is found on our decomposition, then the whole graph really contains an out-branching of the same number of leaves (constructively).

!!

$\Longrightarrow$ There is an XP algorithm for the MinLOB problem on digraphs of bounded clique-width, ... but it does not fit into the dynamic iterative Myhill–Nerode-based scheme. Why?

Final Questions (and remarks)

How to formulate our main question?

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

Particularly for MinLOB, is there any nice canonical partition consistent with $\otimes$? (polynomially bounded!)

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

Particularly for MinLOB, is there any nice canonical partition consistent with $\otimes$? (polynomially bounded!)

Even in finite-index case, one can create an inconsistent canonical partition, but then this is “repaired” by automaton minimization.

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

Particularly for MinLOB, is there any nice canonical partition consistent with $\otimes$? (polynomially bounded!)

Even in finite-index case, one can create an inconsistent canonical partition, but then this is "repaired" by automaton minimization.

In our XP case, what would be the effect of "state minimization"?

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

Particularly for MinLOB, is there any nice canonical partition consistent with $\otimes$? (polynomially bounded!)

Even in finite-index case, one can create an inconsistent canonical partition, but then this is "repaired" by automaton minimization.

In our XP case, what would be the effect of "state minimization"?

So, which one is more likely to be true?

- One can always find an (asymptotically?) optimal canonical partition consistent with $\otimes$.

Final Questions (and remarks)

How to formulate our main question?

Can always (even for unbounded index) Myhill–Nerode be turned into a dynamic algorithm. . . ? (by tabulation)

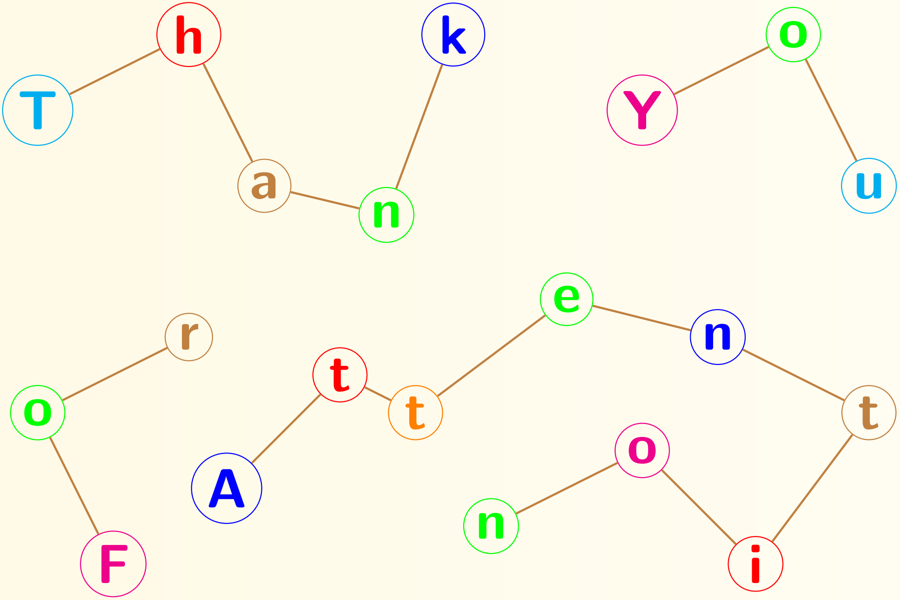
Particularly for MinLOB, is there any nice canonical partition consistent with $\otimes$? (polynomially bounded!)

Even in finite-index case, one can create an inconsistent canonical partition, but then this is "repaired" by automaton minimization.

In our XP case, what would be the effect of "state minimization"?

So, which one is more likely to be true?

- One can always find an (asymptotically?) optimal canonical partition consistent with $\otimes$.
- Or, there is something mysterious going on with Myhill–Nerode for XP algorithms.


Thank
You
for
Attention