Mining Tree Patterns with Partially Injective Homomorphisms

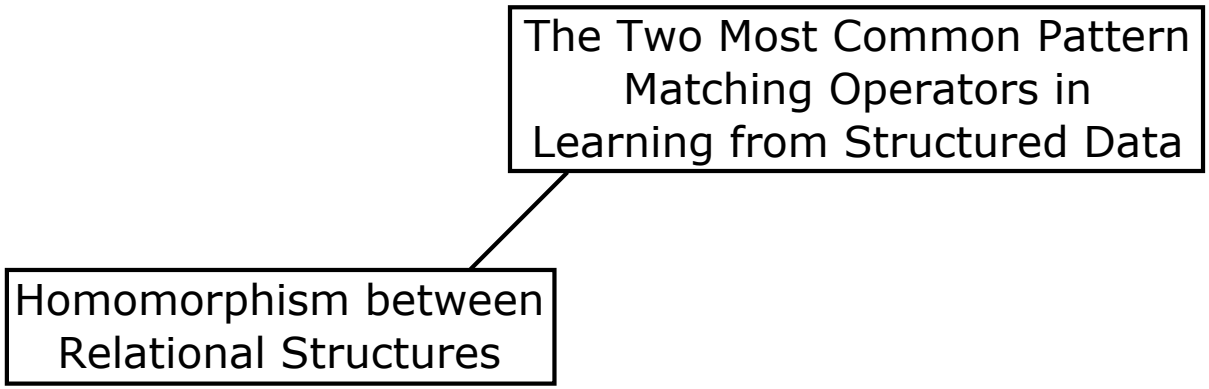
Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel

University of Bonn
Fraunhofer IAIS
Fraunhofer Center for Machine Learning

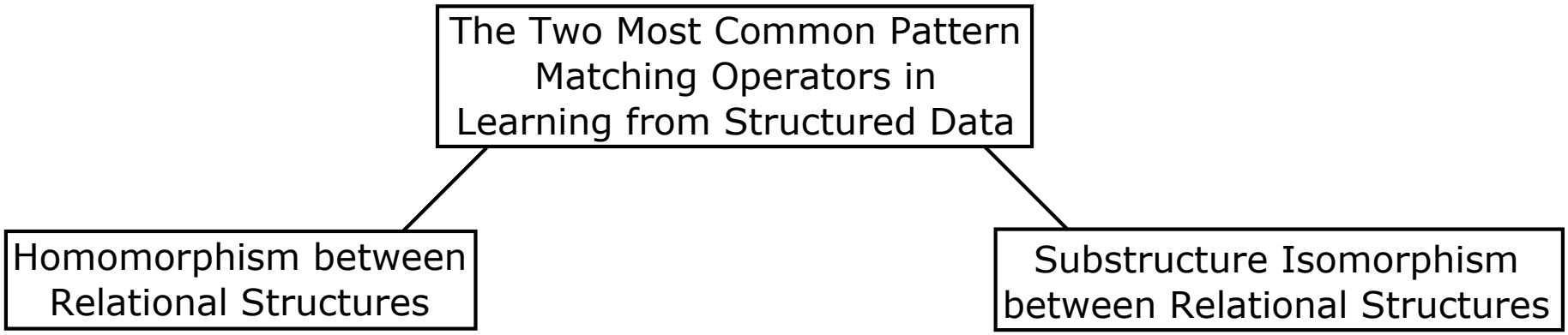
Learning from Structured Data

The Two Most Common Pattern Matching Operators in Learning from Structured Data

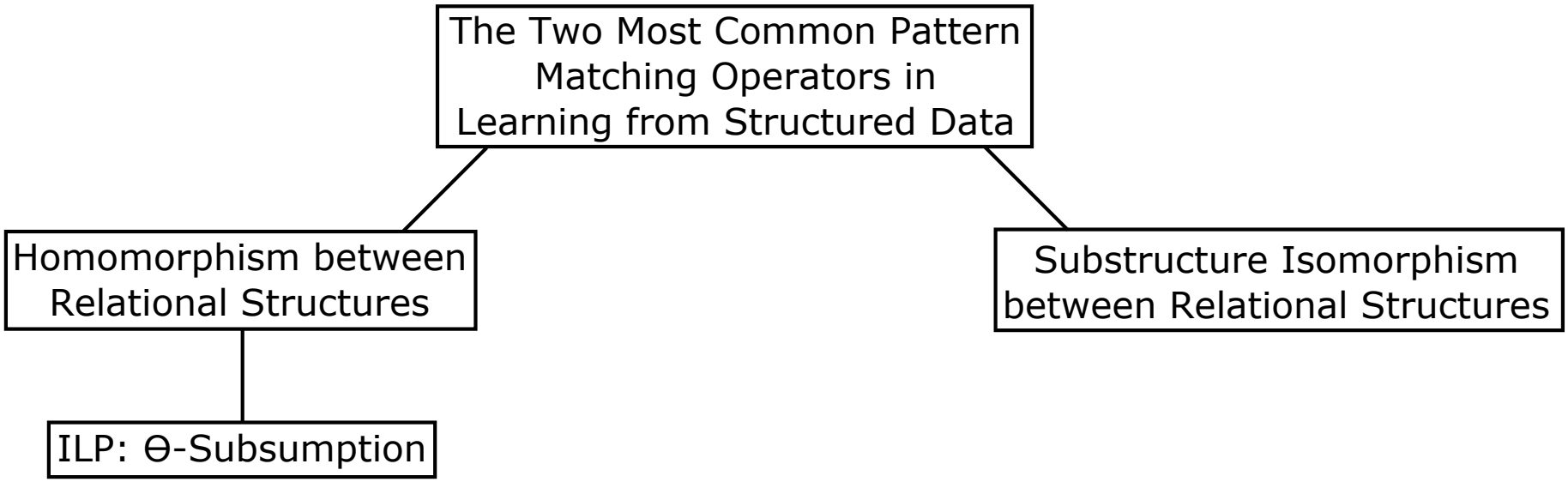
Learning from Structured Data



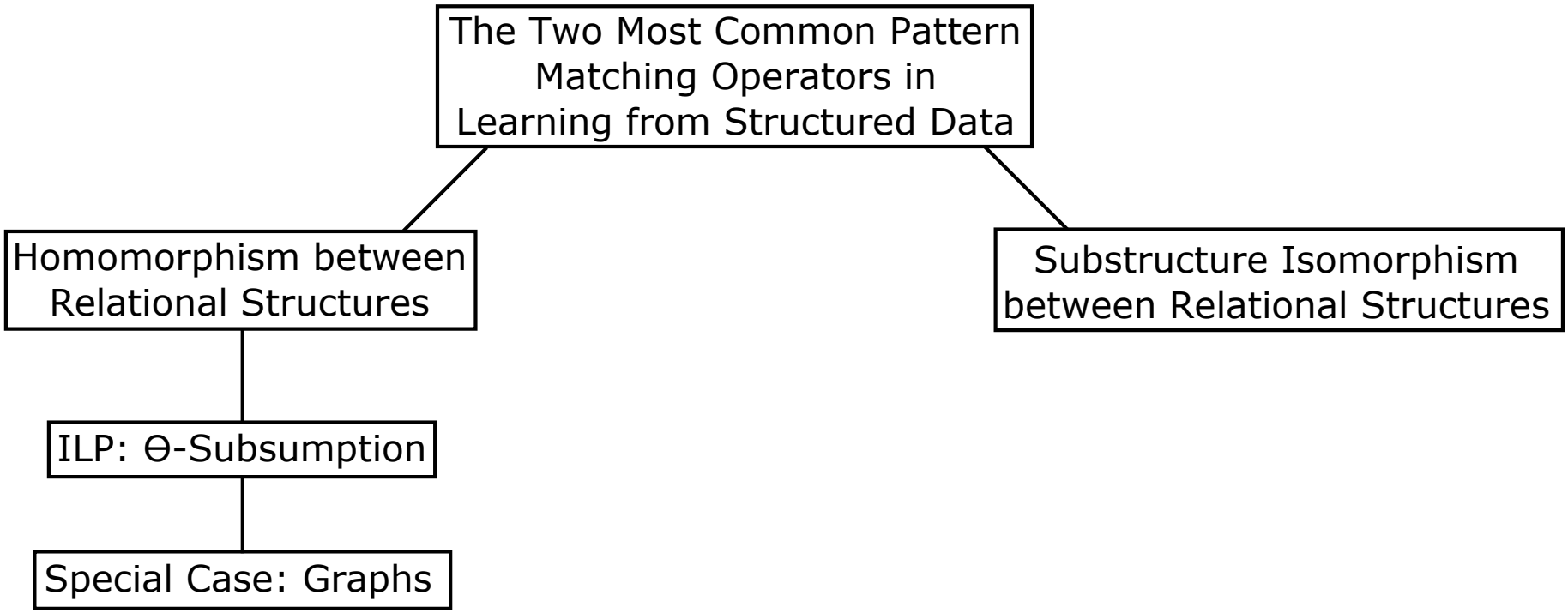
Learning from Structured Data



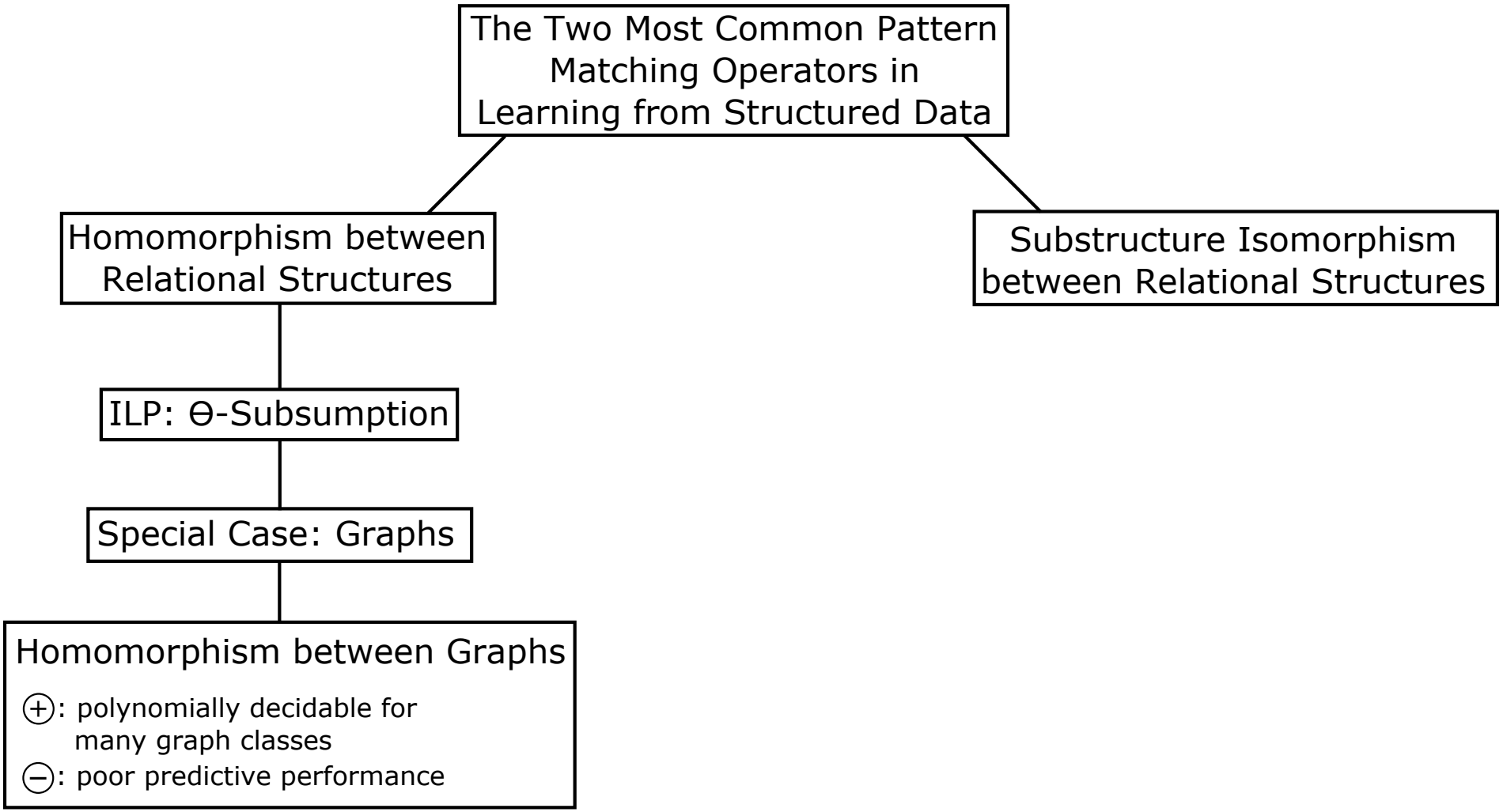
Learning from Structured Data



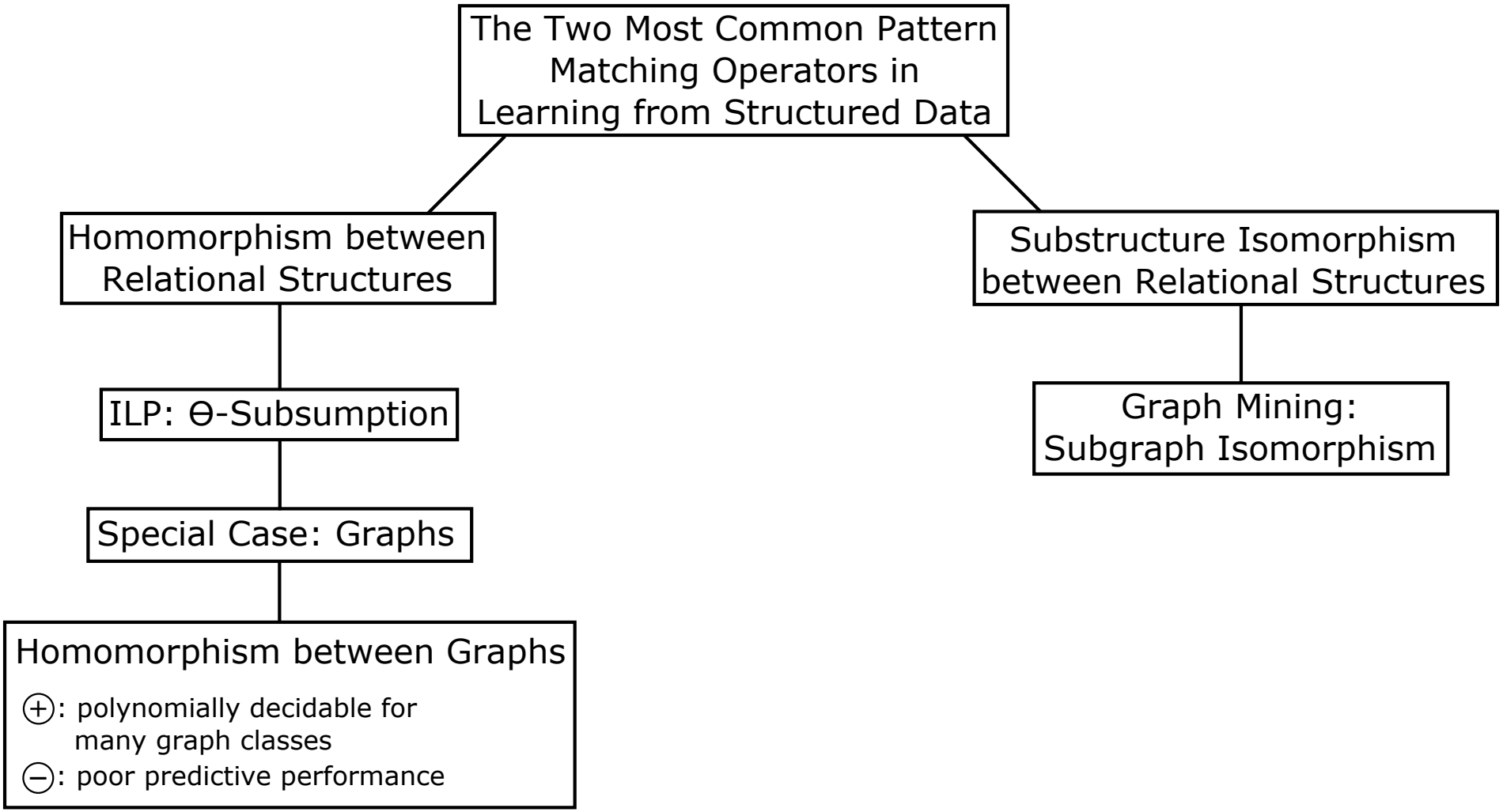
Learning from Structured Data



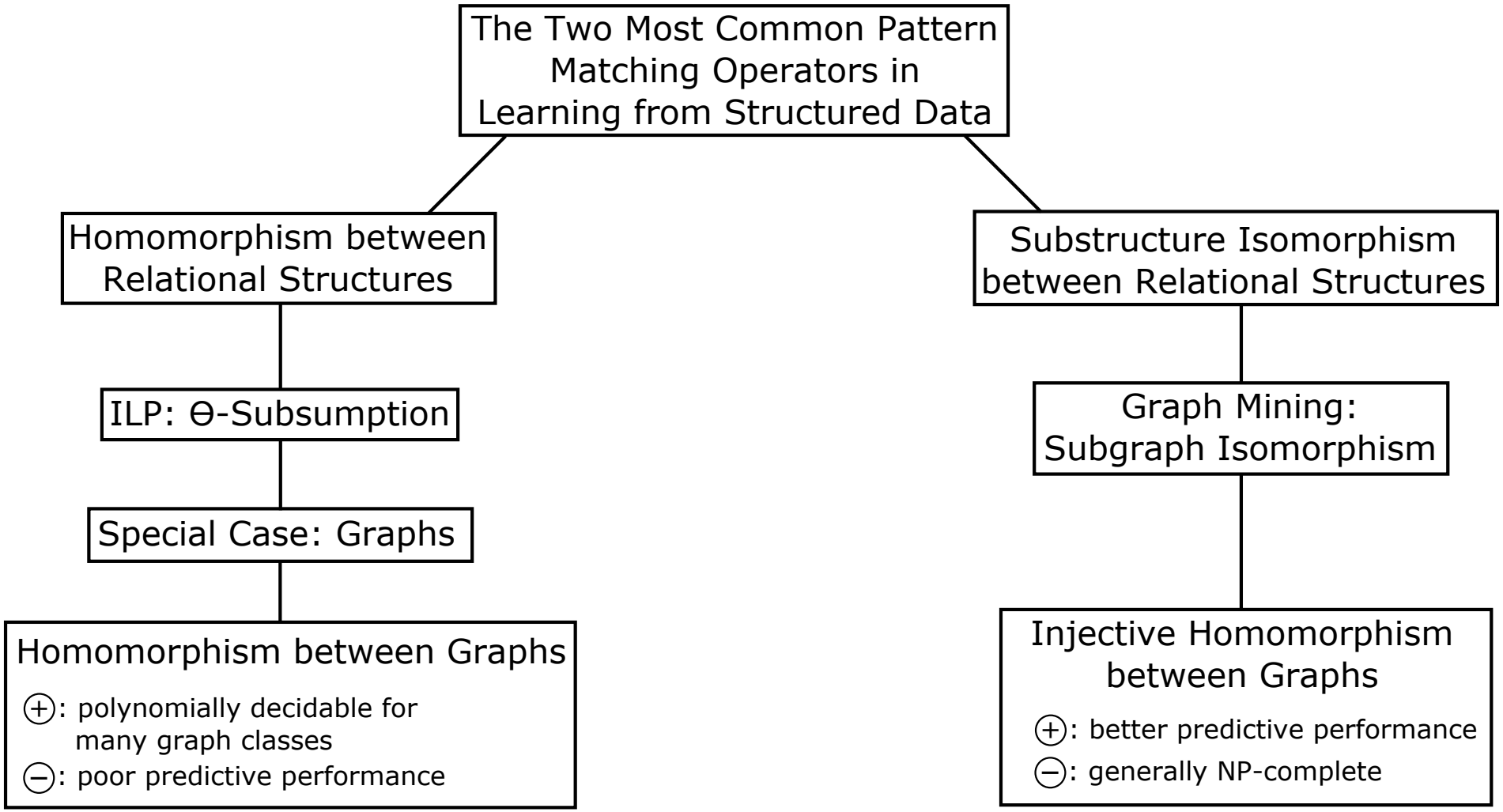
Learning from Structured Data



Learning from Structured Data



Learning from Structured Data



Learning from Structured Data

The Two Most Common Pattern Matching Operators in Learning from Structured Data

Homomorphism between Relational Structures

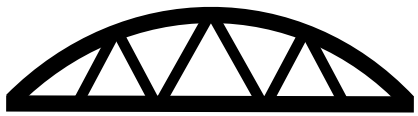
ILP: Θ-Subsumption

Special Case: Graphs

Homomorphism between Graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance



Partially Injective Homomorphism

Substructure Isomorphism between Relational Structures

Graph Mining: Subgraph Isomorphism

Injective Homomorphism between Graphs

⊕: better predictive performance

⊖: generally NP-complete

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

In this work, pattern graphs are restricted to trees

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

In this work, pattern graphs are restricted to trees

- homomorphism from trees is decidable in polynomial time

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

In this work, pattern graphs are restricted to trees

- homomorphism from trees is decidable in polynomial time
- deciding subgraph-isomorphism from trees is NP-complete

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

In this work, pattern graphs are restricted to trees

- homomorphism from trees is decidable in polynomial time
- deciding subgraph-isomorphism from trees is NP-complete

Clearly, the injectivity constraint causes the complexity difference

Goal

Homomorphism between graphs

⊕: polynomially decidable for many graph classes

⊖: poor predictive performance

Partially Injective Homomorphism

Injective Homomorphism between graphs

⊕: better predictive performance

⊖: generally NP-complete

We would like to combine the advantages of both sides

In this work, pattern graphs are restricted to trees

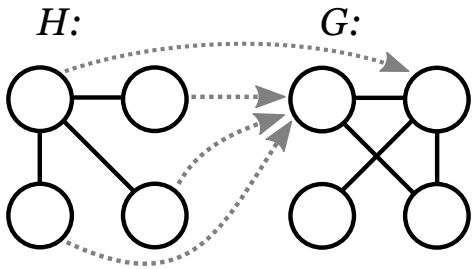
- homomorphism from trees is decidable in polynomial time
- deciding subgraph-isomorphism from trees is NP-complete

Clearly, the injectivity constraint causes the complexity difference

Solution: Bridge the gap by requiring only partial injectivity

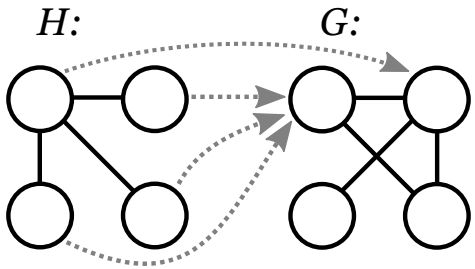
Injective Homomorphism

Graph homomorphism:

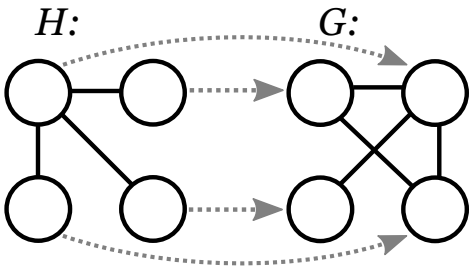


Injective Homomorphism

Graph homomorphism:

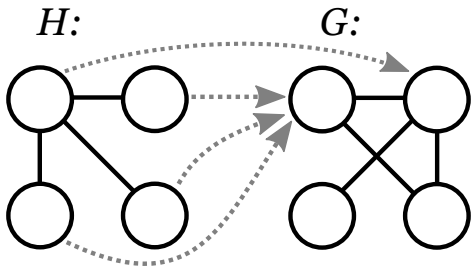


Subgraph isomorphism can be reduced to homomorphism:

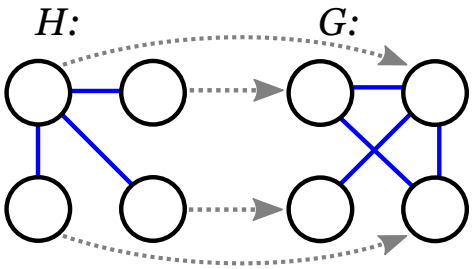


Injective Homomorphism

Graph homomorphism:



Subgraph isomorphism can be reduced to homomorphism:

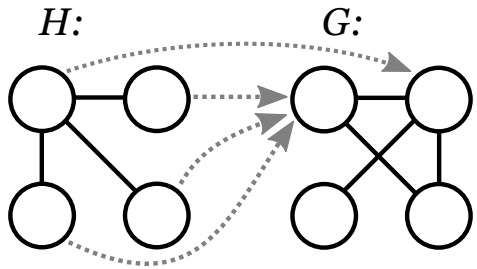


Transform graphs H, G into H', G', respectively, as follows:

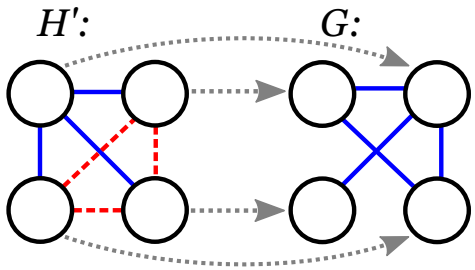
- Color all edges of H and G in blue (original edges)

Injective Homomorphism

Graph homomorphism:



Subgraph isomorphism can be reduced to homomorphism:

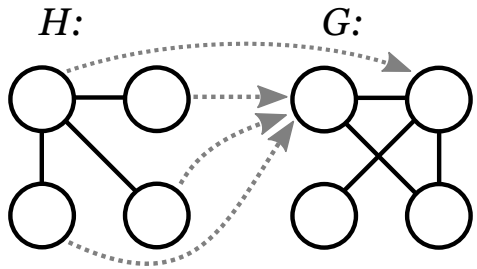


Transform graphs H, G into H', G', respectively, as follows:

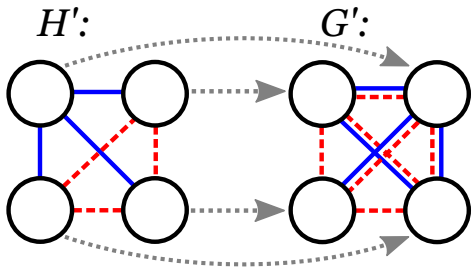
- Color all edges of H and G in blue (original edges)
- Add to H red edges between all *unconnected* vertex pairs (constraint edges)

Injective Homomorphism

Graph homomorphism:



Subgraph isomorphism can be reduced to homomorphism:

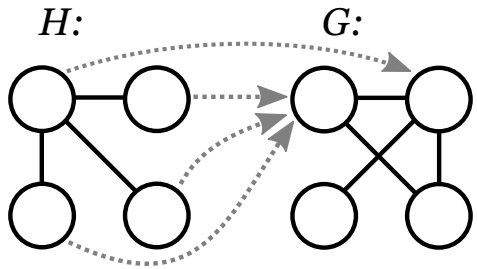


Transform graphs H, G into H', G', respectively, as follows:

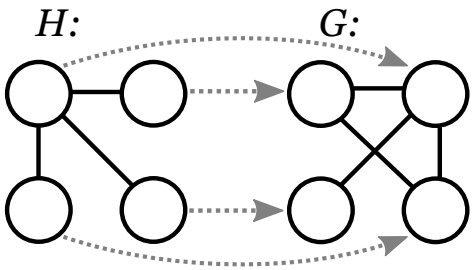
- Color all edges of H and G in blue (original edges)
- Add to H red edges between all *unconnected* vertex pairs (constraint edges)
- Connect *all* vertex pairs in G by a red edge

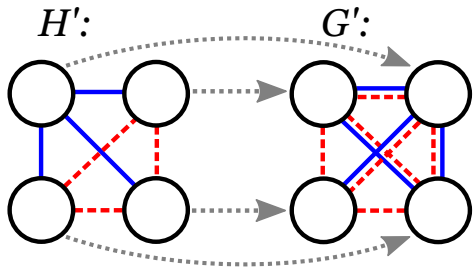
Injective Homomorphism

Graph homomorphism:



Subgraph isomorphism can be reduced to homomorphism:





Transform graphs H, G into H', G', respectively, as follows:

- Color all edges of H and G in blue (original edges)
- Add to H red edges between all *unconnected* vertex pairs (constraint edges)
- Connect *all* vertex pairs in G by a red edge

H is subgraph isomorphic to G iff there exists a homomorphism from H' into G'

Partially Injective Homomorphism

Partially injective homomorphism requires injectivity constraints for only a subset of vertex pairs in the pattern

Partially Injective Homomorphism

Partially injective homomorphism requires injectivity constraints for only a subset of vertex pairs in the pattern

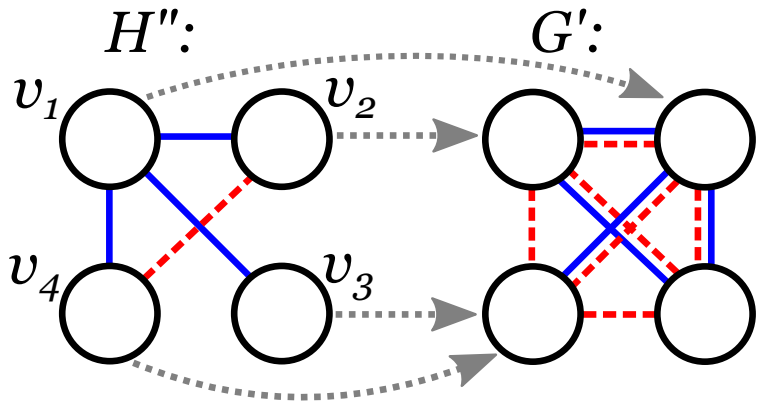
$\Rightarrow$ i.e. add to H only a selection of red edges

Partially Injective Homomorphism

Partially injective homomorphism requires injectivity constraints for only a subset of vertex pairs in the pattern

$\Rightarrow$ i.e. add to H only a selection of red edges

Example:

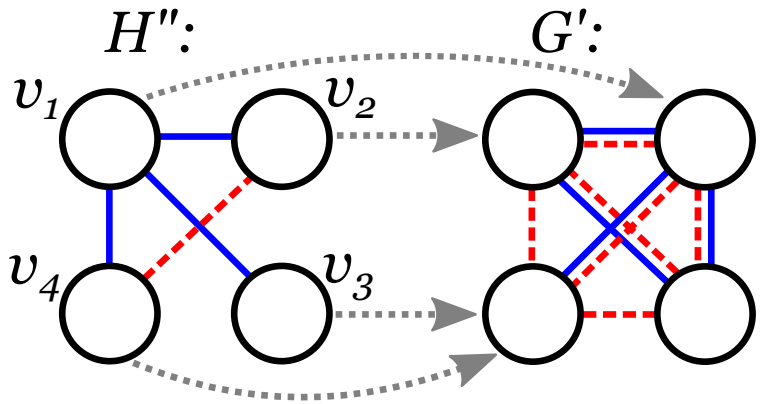


Partially Injective Homomorphism

Partially injective homomorphism requires injectivity constraints for only a subset of vertex pairs in the pattern

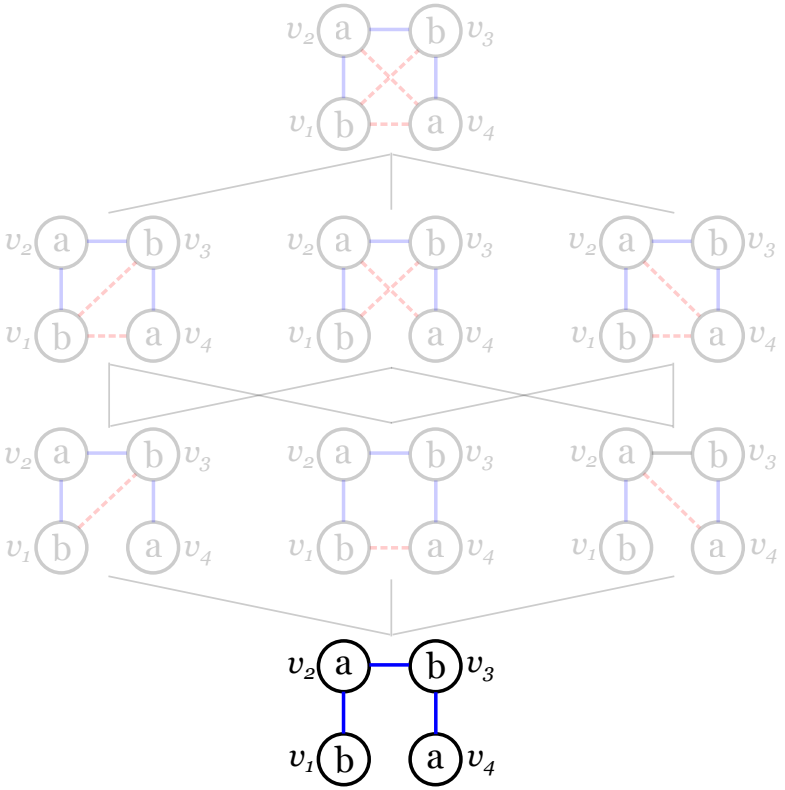
$\Rightarrow$ i.e. add to H only a selection of red edges

Example:



v_2 and v_4 of H'' must be mapped to distinct vertices in G'

Main Idea



Complexity

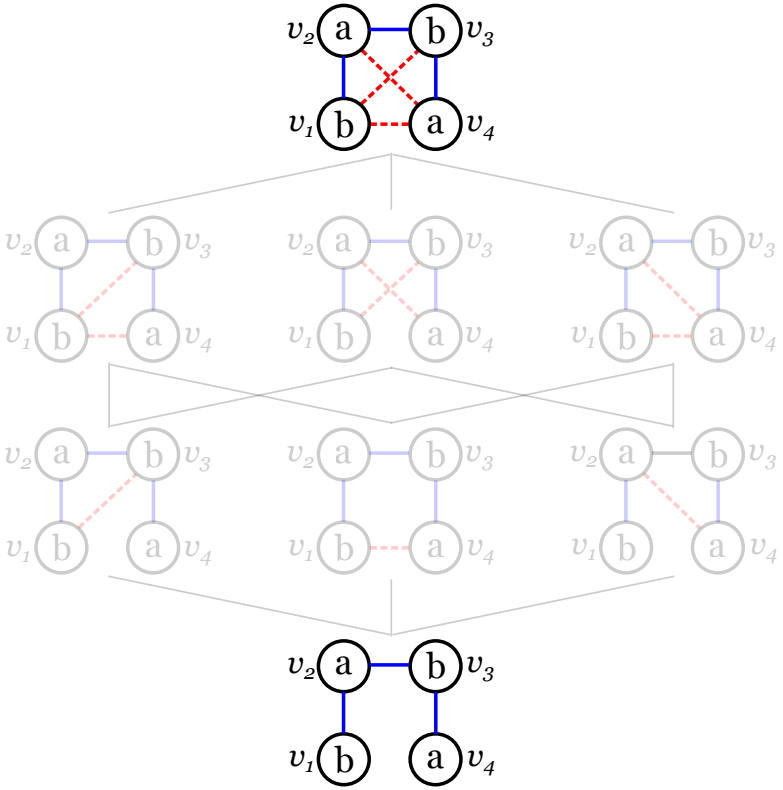
subgraph-isomorphism

partial injective homorphisms with two red edges

partial injective homorphisms with one red edge

homomorphism

Main Idea



Complexity

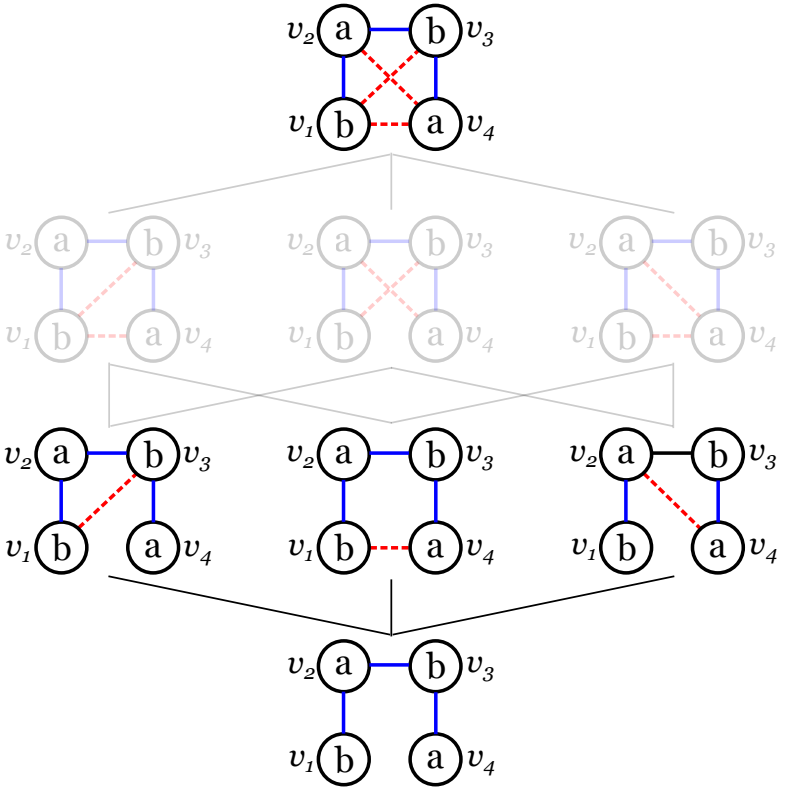
subgraph-isomorphism

partial injective homorphisms with two red edges

partial injective homorphisms with one red edge

homomorphism

Main Idea



Complexity

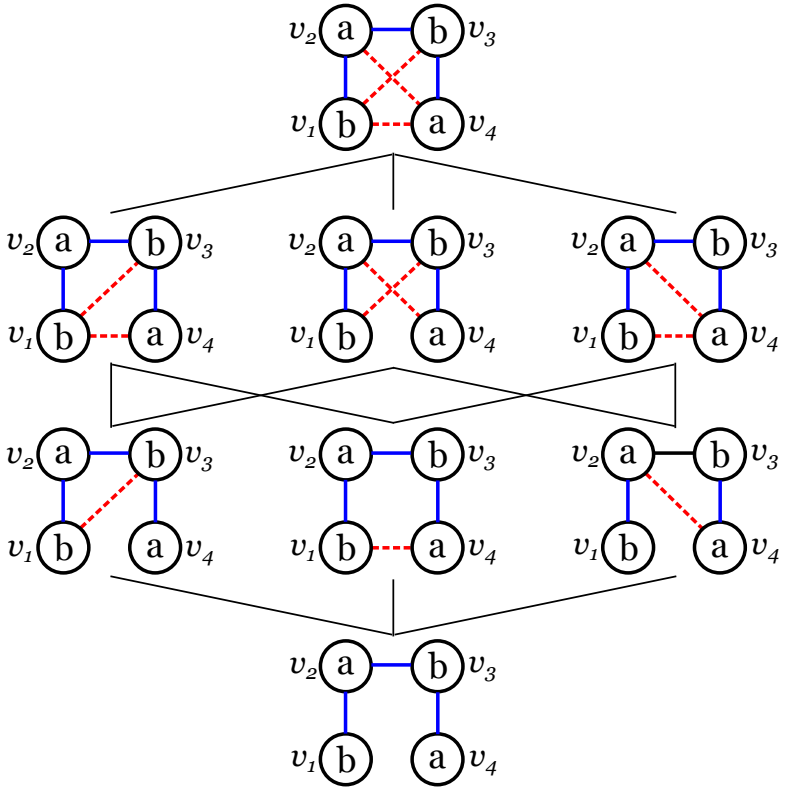
subgraph-isomorphism

partial injective homorphisms with two red edges

partial injective homorphisms with one red edge

homomorphism

Main Idea



Complexity

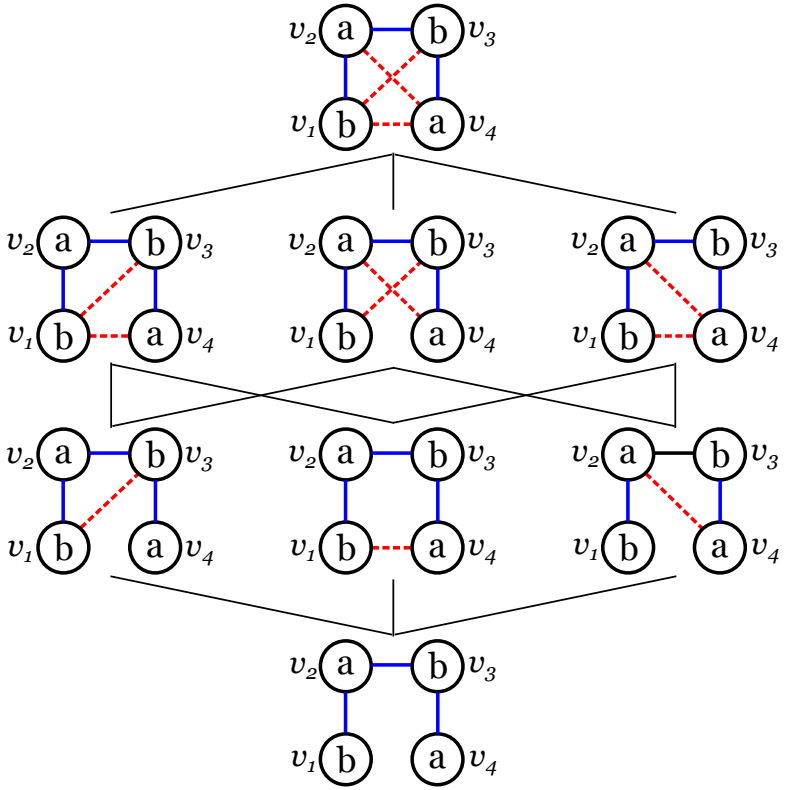
subgraph-isomorphism

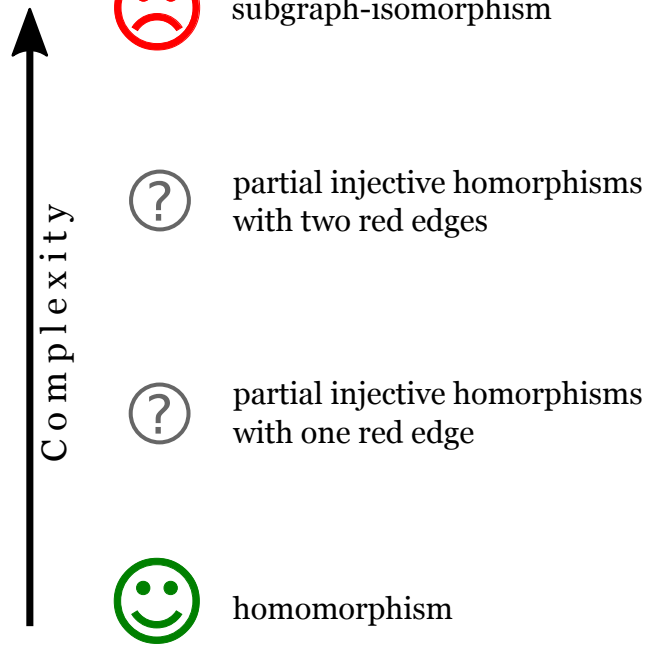
partial injective homorphisms with two red edges

partial injective homorphisms with one red edge

homomorphism

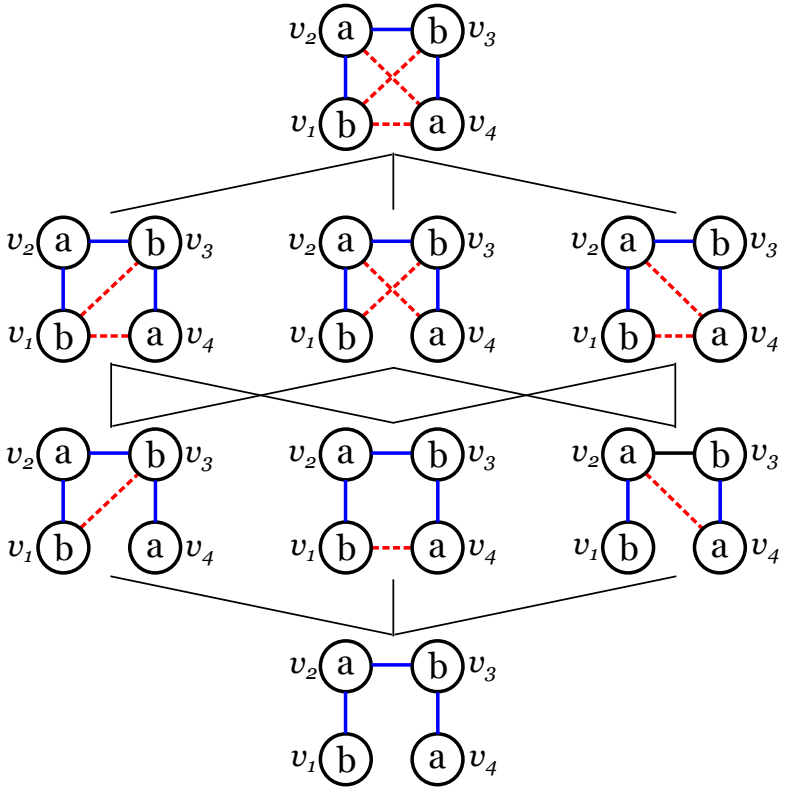
Main Idea

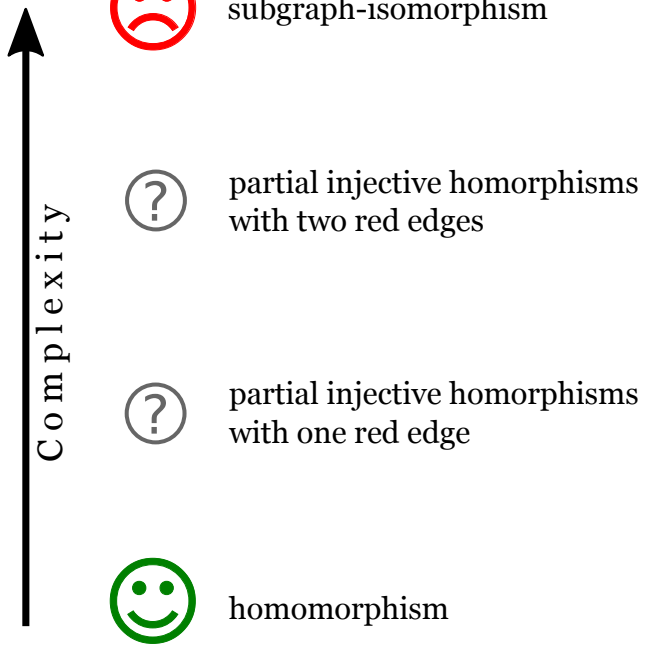




☺ **Partially injective homomorphism can be decided in polynomial time if the pattern graph (blue + red edges) has bounded tree-width.**

Main Idea



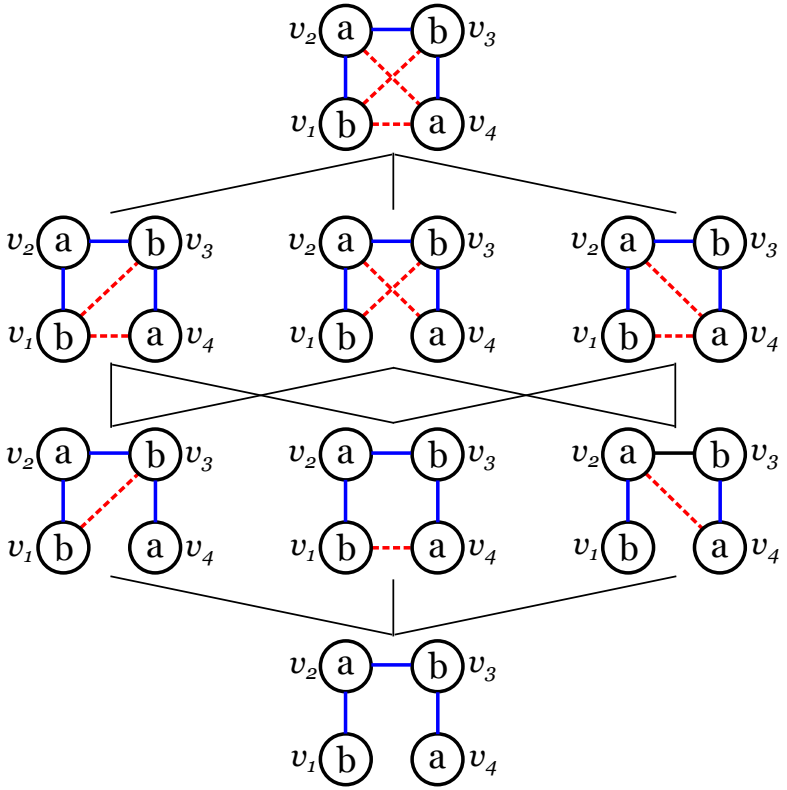


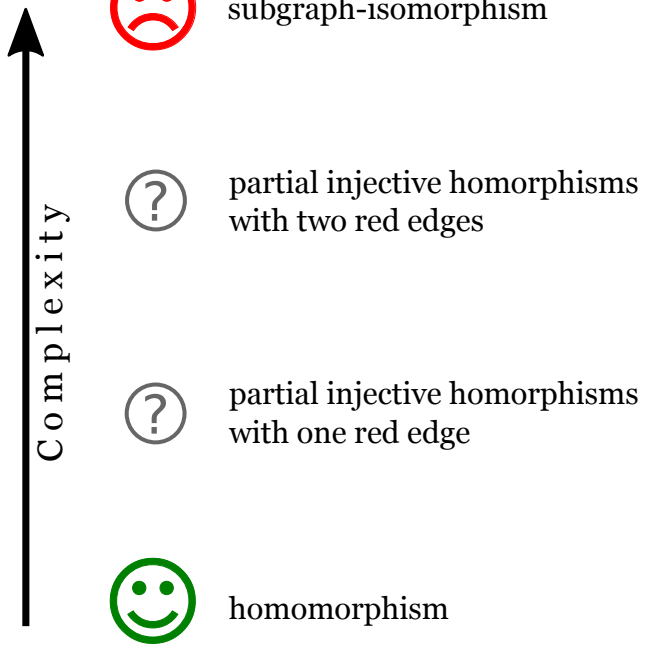
Partially injective homomorphism can be decided in polynomial time if the pattern graph (blue + red edges) has bounded tree-width.

Tree-width:

- Positive integer measuring the "tree-likeness"

Main Idea



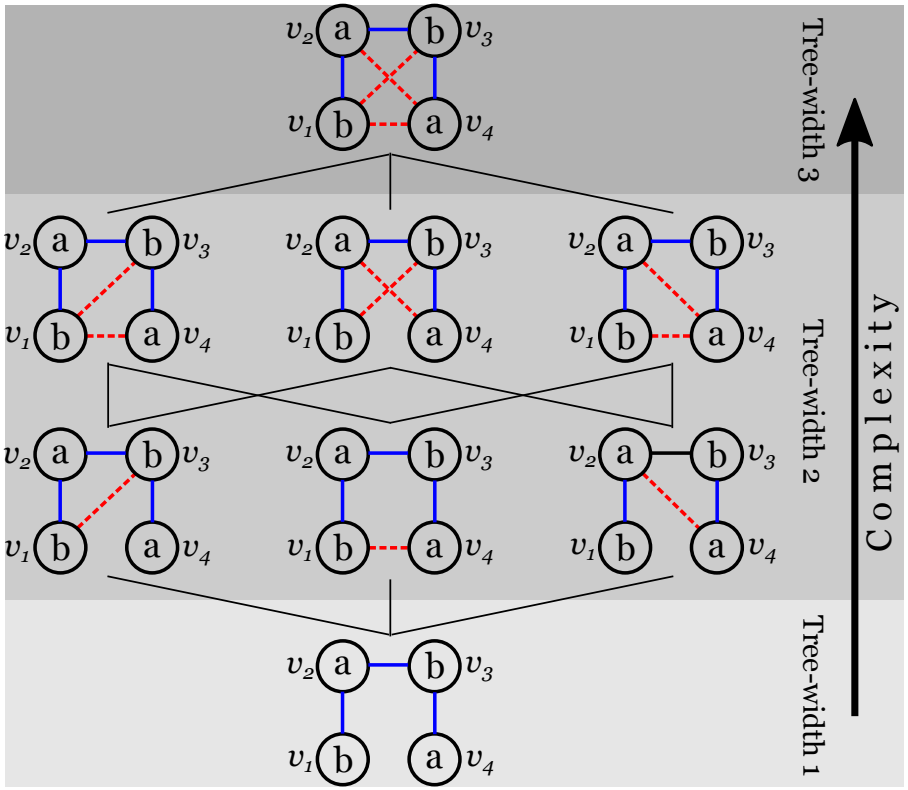


☺ **Partially injective homomorphism can be decided in polynomial time if the pattern graph (blue + red edges) has bounded tree-width.**

Tree-width:

- Positive integer measuring the "tree-likeness"
- Generally intractable problems become tractable for constant tree-width

Main Idea



subgraph-isomorphism

partial injective homorphisms with two red edges

partial injective homorphisms with one red edge

homomorphism

☺ **Partially injective homomorphism can be decided in polynomial time if the pattern graph (blue + red edges) has bounded tree-width.**

Tree-width:

- Positive integer measuring the "tree-likeness"
- Generally intractable problems become tractable for constant tree-width

Our approach

Generate *frequent trees* w.r.t. partially injective homomorphism

Our approach

Generate *frequent trees* w.r.t. partially injective homomorphism

- i.e. original (blue) edges form a tree

Our approach

Generate *frequent trees* w.r.t. partially injective homomorphism

- i.e. original (blue) edges form a tree
- and together with constraints (red egdes) form a graph of bounded tree-width

Our approach

Generate *frequent trees* w.r.t. partially injective homomorphism

- i.e. original (blue) edges form a tree
- and together with constraints (red egdes) form a graph of bounded tree-width

Mining algorithm lists *all* frequent patterns up to a user-defined size in *incremental polynomial time*.

Our approach

Generate *frequent trees* w.r.t. partially injective homomorphism

- i.e. original (blue) edges form a tree
- and together with constraints (red egdes) form a graph of bounded tree-width

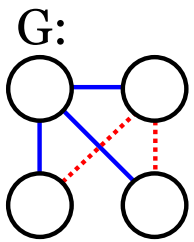
Mining algorithm lists *all* frequent patterns up to a user-defined size in *incremental polynomial time*.

Pattern generation is based on refinement operator.

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

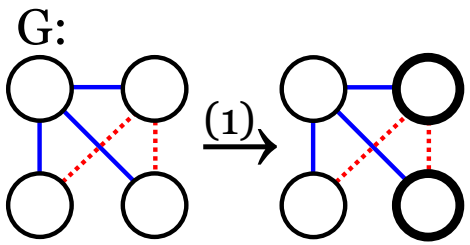
Idea for tree-width $k = 2$:



Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:

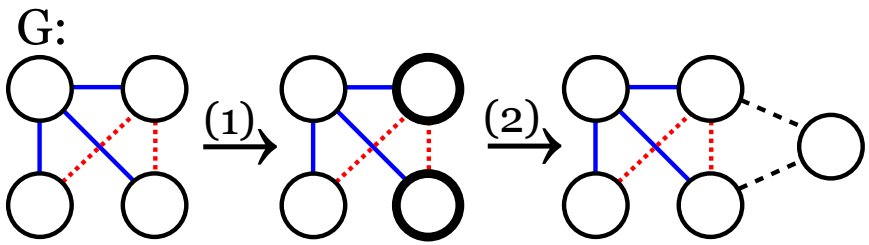


(1) select a 2-clique

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



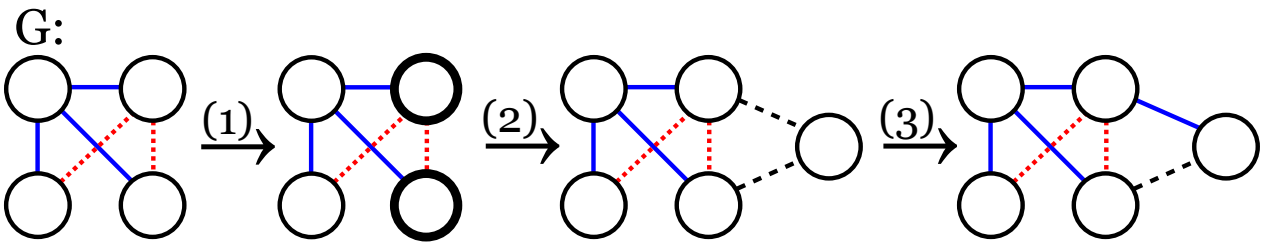
(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



(1) select a 2-clique

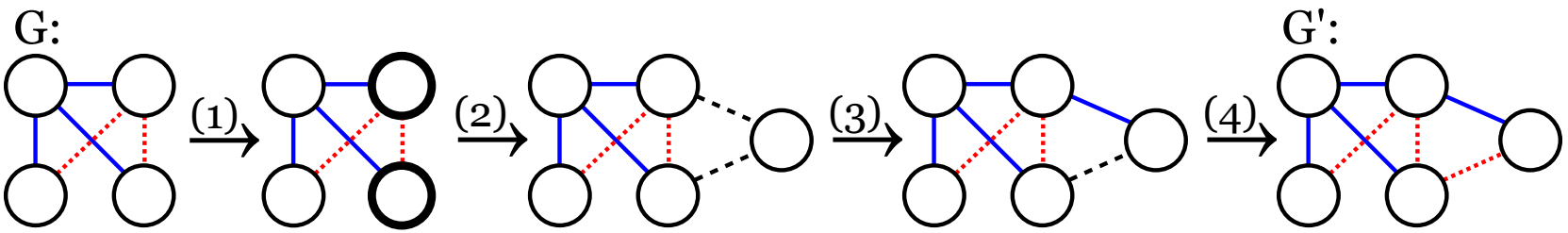
(2) add a vertex and connect it to the 2-clique

(3) color one edge blue

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

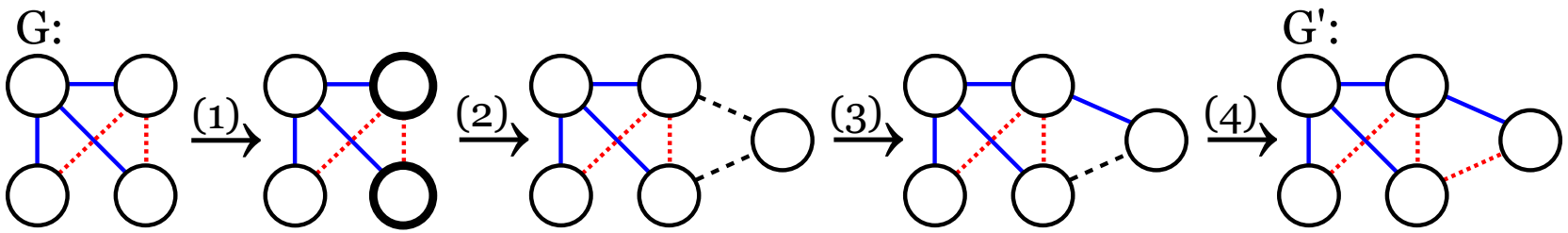
(3) color one edge blue

(4) color all others red

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

(3) color one edge blue

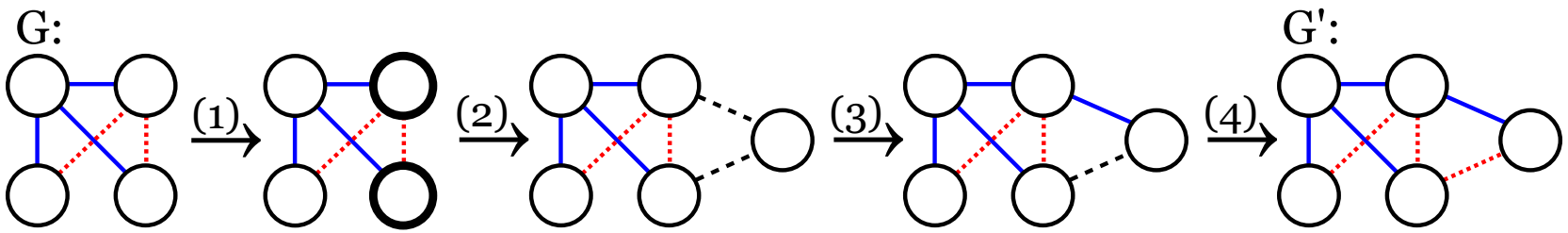
(4) color all others red

G' is a refinement of G. Both graphs are k-trees with tree-width 2.

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

(3) color one edge blue

(4) color all others red

G' is a refinement of G. Both graphs are k-trees with tree-width 2.

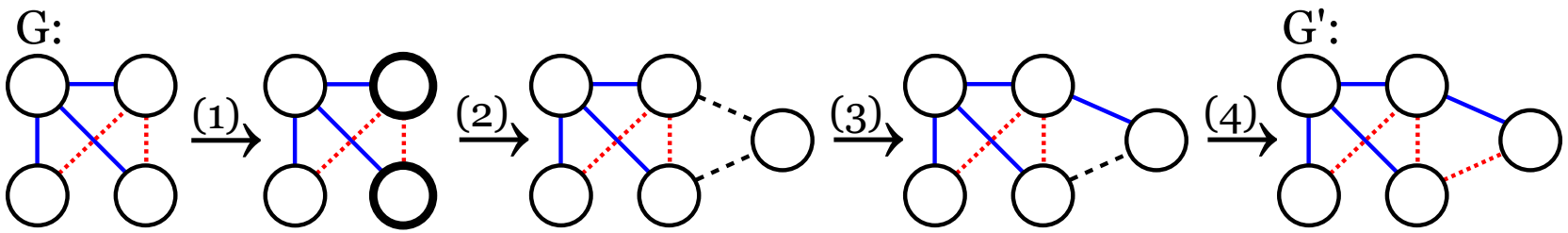
Properties:

- graphs are *maximally* constrained (i.e. adding another red edge increases the tree-width)

Refinement Operator

Refinement step (utilizes definition of *k-trees*):

Idea for tree-width $k = 2$:



(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

(3) color one edge blue

(4) color all others red

G' is a refinement of G. Both graphs are k-trees with tree-width 2.

Properties:

- graphs are *maximally* constrained (i.e. adding another red edge increases the tree-width)
- the embedding decision problem is guaranteed to lie in P

Evaluation Setup

We performed prediction tasks over molecular data

Evaluation Setup

We performed prediction tasks over molecular data

Each molecule represented as a graph has a binary label

Evaluation Setup

We performed prediction tasks over molecular data

Each molecule represented as a graph has a binary label

SVMs were used to separate points representing molecular graphs in feature space spanned by patterns

Experiments I: Predictive Performance

Q: How does the predictive performance using patterns w.r.t. partially injective homomorphism compare to patterns generated w.r.t. subgraph isomorphism?

Experiments I: Predictive Performance

Q: How does the predictive performance using patterns w.r.t. partially injective homomorphism compare to patterns generated w.r.t. subgraph isomorphism?

Frequent Patterns	MUTAG	PTC	NCI1	NCI109
s.g.i. graphs	91.99 ± 6.65	73.07 ± 9.34	89.33 ± 1.13	88.55 ± 1.77
s.g.i. trees	91.63 ± 5.89	73.08 ± 9.39	89.14 ± 1.19	88.37 ± 1.72
p.i.h. trees ($k = 4$)	90.49 ± 6.98	72.38 ± 8.02	88.30 ± 1.29	87.54 ± 2.03
p.i.h. trees ($k = 3$)	90.21 ± 8.36	73.24 ± 8.55	88.77 ± 1.32	87.77 ± 2.02
p.i.h. trees ($k = 2$)	76.92 ± 14.90	67.84 ± 6.53	87.68 ± 1.35	86.72 ± 1.66

Prediction measures stated as AUC values in % for different tree-width choices k in contrast to freq. subgraphs and subtrees.

Experiments I: Predictive Performance

Q: How does the predictive performance using patterns w.r.t. partially injective homomorphism compare to patterns generated w.r.t. subgraph isomorphism?

Frequent Patterns	MUTAG	PTC	NCI1	NCI109
s.g.i. graphs	91.99 ± 6.65	73.07 ± 9.34	89.33 ± 1.13	88.55 ± 1.77
s.g.i. trees	91.63 ± 5.89	73.08 ± 9.39	89.14 ± 1.19	88.37 ± 1.72
p.i.h. trees ($k = 4$)	90.49 ± 6.98	72.38 ± 8.02	88.30 ± 1.29	87.54 ± 2.03
p.i.h. trees ($k = 3$)	90.21 ± 8.36	73.24 ± 8.55	88.77 ± 1.32	87.77 ± 2.02
p.i.h. trees ($k = 2$)	76.92 ± 14.90	67.84 ± 6.53	87.68 ± 1.35	86.72 ± 1.66

Prediction measures stated as AUC values in % for different tree-width choices k in contrast to freq. subgraphs and subtrees.

A: For already tree-width $k = 3$, the results are very close to those of ordinary frequent trees (subgraph isomorphism).

Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Setup:

- the set of tree patterns is fixed

Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Setup:

- the set of tree patterns is fixed
- we gradually increased the number of red egdes by increasing the tree-width k

Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Setup:

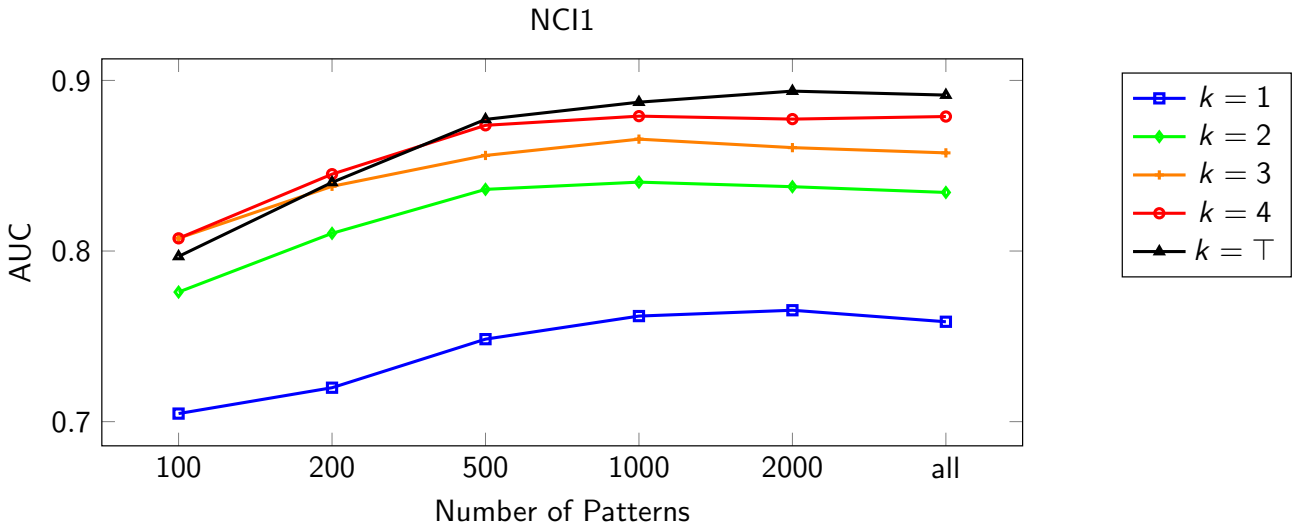
- the set of tree patterns is fixed
- we gradually increased the number of red egdes by increasing the tree-width k
- the degree of injectivity in patterns ranges from $k = 1$ (ordinary homomorphism) to $k = \top$ (subgraph isomorphism)

Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Setup:

- the set of tree patterns is fixed
- we gradually increased the number of red egdes by increasing the tree-width k
- the degree of injectivity in patterns ranges from $k = 1$ (ordinary homomorphism) to $k = \top$ (subgraph isomorphism)

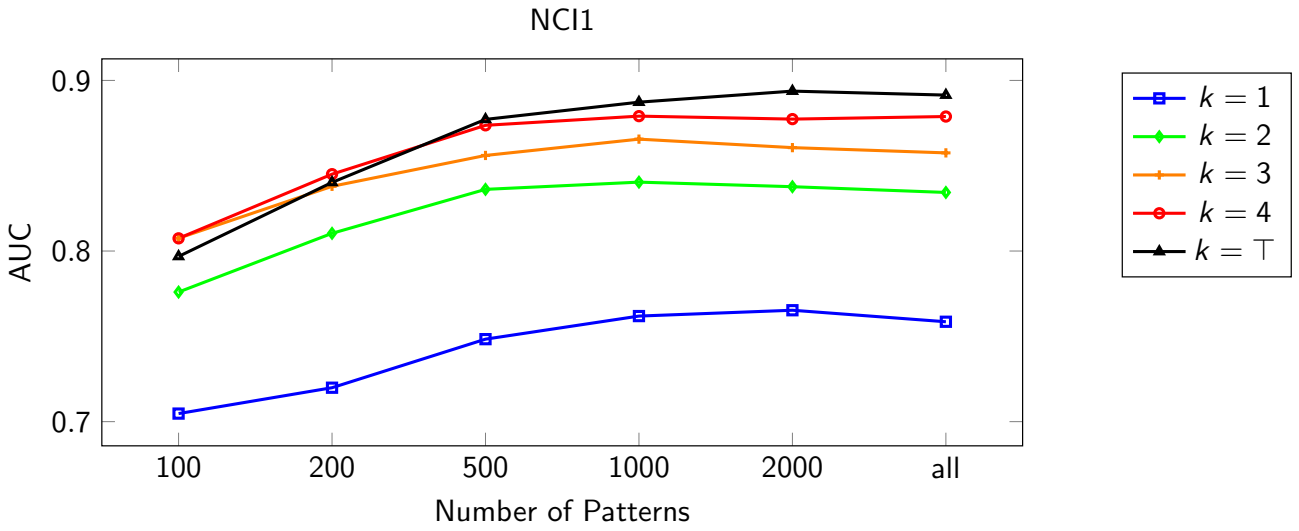


Experiments II: Injectivity Degree

Q: How does the degree of injectivity in the pattern matching operator influence the predictive performance?

Setup:

- the set of tree patterns is fixed
- we gradually increased the number of red egdes by increasing the tree-width k
- the degree of injectivity in patterns ranges from $k = 1$ (ordinary homomorphism) to $k = \top$ (subgraph isomorphism)



A: The predictive performance scales with the degree of injectivity of the underlying pattern matching operator.

Experiments III: Runtimes

Q: How do runtimes of our algorithm compare to subgraph-isomorphism-based graph miners?

Experiments III: Runtimes

Q: How do runtimes of our algorithm compare to subgraph-isomorphism-based graph miners?

	MUTAG	PTC	NCI1	NCI109	Erdős-Rényi random graphs (unlabeled)			
					$q = 1.0$	$q = 1.5$	$q = 2.0$	$q = 3.0$
GASTON (EL)	0.1	0.2	2.6	2.7	2.8	54.6	*mem err*	*mem err*
GASTON (RE)	0.3	1.0	8.5	8.6	5.0	39.5	1163.0	31061.4
FSG	0.7	4.1	30.2	29.9	194.2	10584.9	10888.4	10852.5
PIH Miner ($k = 3$)	0.8	3.4	27.3	24.1	0.3	1.3	2.8	8.4

Runtimes (in sec.) of our algorithm in comparison to GASTON and FSG on molecular and artificial datasets.

Experiments III: Runtimes

Q: How do runtimes of our algorithm compare to subgraph-isomorphism-based graph miners?

	MUTAG	PTC	NCI1	NCI109	Erdős-Rényi random graphs (unlabeled)			
					$q = 1.0$	$q = 1.5$	$q = 2.0$	$q = 3.0$
GASTON (EL)	0.1	0.2	2.6	2.7	2.8	54.6	*mem err*	*mem err*
GASTON (RE)	0.3	1.0	8.5	8.6	5.0	39.5	1163.0	31061.4
FSG	0.7	4.1	30.2	29.9	194.2	10584.9	10888.4	10852.5
PIH Miner ($k = 3$)	0.8	3.4	27.3	24.1	0.3	1.3	2.8	8.4

Runtimes (in sec.) of our algorithm in comparison to GASTON and FSG on molecular and artificial datasets.

A: While our algorithm is slower on real-world datasets, it is much faster on (unlabeled) graphs of a higher structural complexity (i.e., higher value q).

Summary

Partially Injective Homomorphisms:

- a new kind of pattern matching operators

Summary

Partially Injective Homomorphisms:

- a new kind of pattern matching operators
- bridges the gap between homomorphism and subgraph isomorphism

Summary

Partially Injective Homomorphisms:

- a new kind of pattern matching operators
- bridges the gap between homomorphism and subgraph isomorphism
- dynamic pattern matching

Summary

Partially Injective Homomorphisms:

- a new kind of pattern matching operators
- bridges the gap between homomorphism and subgraph isomorphism
- dynamic pattern matching

In this work:

- efficiency is guaranteed by bounded tree-width graphs

Summary

Partially Injective Homomorphisms:

- a new kind of pattern matching operators
- bridges the gap between homomorphism and subgraph isomorphism
- dynamic pattern matching

In this work:

- efficiency is guaranteed by bounded tree-width graphs
- experimental results show that patterns do not lose much expressiveness (compared to subgraph isomorphism)

Summary

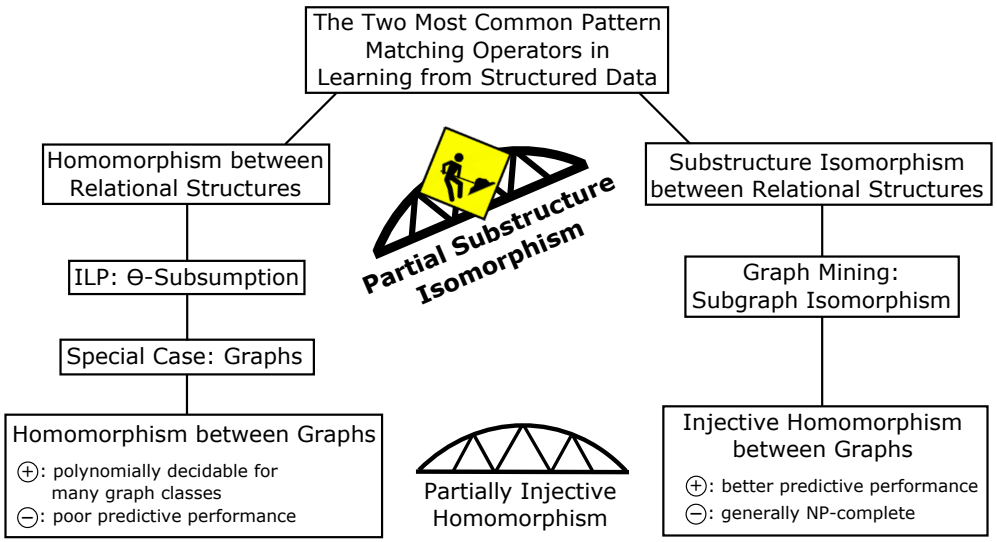
Partially Injective Homomorphisms:

- a new kind of pattern matching operators
- bridges the gap between homomorphism and subgraph isomorphism
- dynamic pattern matching

In this work:

- efficiency is guaranteed by bounded tree-width graphs
- experimental results show that patterns do not lose much expressiveness (compared to subgraph isomorphism)

Upcoming work:

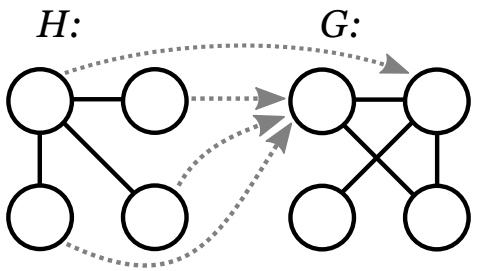


Appendix

Homomorphism & Subgraph Isomorphism

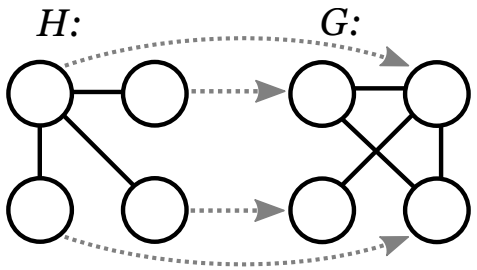
A **homomorphism** from a graph H (the *pattern*) into a graph G (the *text*) is a mapping $\varphi : V(H) \to V(G)$ that preserves the edges (i.e., $uv \in E(H)$ implies $\varphi(u)\varphi(v) \in E(G)$ for all $u, v \in V(H)$).

Example:



A **subgraph isomorphism** from H to G is an *injective* homomorphism $\psi : V(H) \to V(G)$ (i.e. ψ is a homomorphism from H into G and for all $u, v \in V(H)$ with $u \neq v$ holds $\psi(u) \neq \psi(v)$).

Example:



Tree-width

A **tree decomposition** of a graph $G = (V, E)$ is a pair $TD(G) = (T, X)$ where

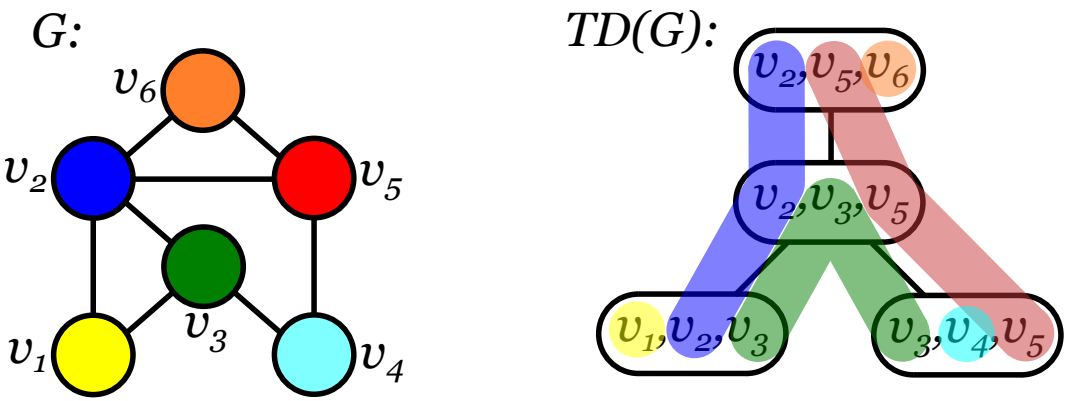
- $T = (I, F)$ is an unordered tree,
- $X = \{bag(i) : i \in I\}$ is a family of subsets of V, s.t.

  (i) $\bigcup_{i \in I} bag(i) = V$
  (ii) for every $\{u, v\} \in E$ there is an $i \in I$ with $\{u, v\} \subseteq bag(i)$
  (iii) for every $v \in V$ the set of nodes $\{i | v \in bag(i)\}$ forms a subtree of T

The **width** of $TD(G)$ is $max_i |bag(i)| - 1$

The **tree-width** of G is the minimum width over all tree decompositions of G

Example:



$TD(G)$ has a width of 2 which is also the tree-width of G.

k-trees

Algorithmic definition of **k-trees**:

(i) A clique of $k+1$ vertices is a k-tree and

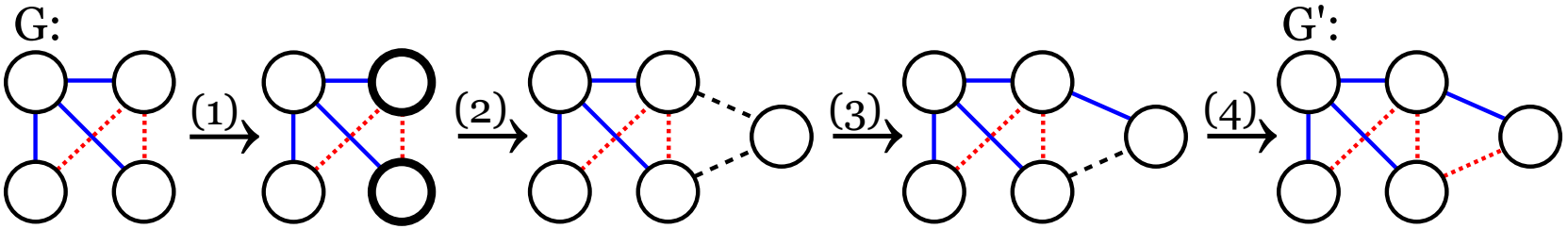
(ii) given a k-tree T_k with n vertices, a k-tree with $n+1$ vertices is obtained from T_k by adding a new vertex v to T_k and connecting v to all vertices of a k-clique of T_k.

Properties:

- A k-tree has tree-width k
- Adding an edge to a $k-$ *tree* results in a graph of tree-width $k+1$.

Refinement Operator

Refinement step:



(1) select a 2-clique

(2) add a vertex and connect it to the 2-clique

(3) color one edge blue

(4) color all others red

G' is a refinement of G. Both graphs are k-trees with $k = 2$.

Properties:

- graphs are *maximally* constrained (i.e. adding another red edge increases the tree-width)
- the embedding decision problem is guaranteed to lie in P

Partial Substructure Isomorphism

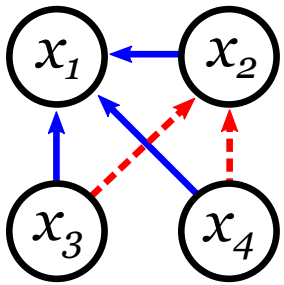
Partially Injective Homomorphisms can be generalized to first-order logic:

Let A and B be function-free first-order clauses. A **partial substructure isomorphism** from A to B satisfying the injectivity constraints in $\mathcal{C} \subseteq [\mathrm{Var}(A)]^2$ is a substitution θ such that $A\theta \subseteq B$ and for all $xy \in \mathcal{C}$, $x\theta \neq y\theta$.

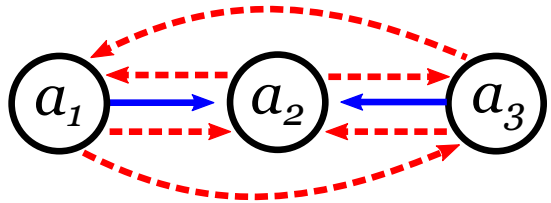
Example:

$$A = \{P(x_2, x_1), P(x_3, x_1), P(x_4, x_1)\}$$
$$B = \{P(a_1, a_2), P(a_3, a_2)\}$$
$$\mathcal{C} = \{x_3x_2, x_4x_2\}$$

A':



B':



There is a partial substructure isomorphism from A to B w.r.t. constraints $\mathcal{C}$ iff A' subsumes B'.