Subgraph Isomorphism Meets Cutting Planes

Jakob Nordström

KTH Royal Institute of Technology

NordConsNet 2019
Simula Research Laboratory
Oslo, Norway
May 21, 2019

Joint work in progress with Jan Elffers, Stephan Gocht, Ciaran McCreesh, . . .

Subgraph Isomorphism Meets Cutting Planes

Jakob Nordström

~~KTH Royal Institute of Technology~~
University of Copenhagen

NordConsNet 2019
Simula Research Laboratory
Oslo, Norway
May 21, 2019

Joint work in progress with Jan Elffers, Stephan Gocht, Ciaran McCreesh, . . .

The Problem

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

The Problem

Input

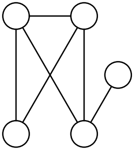
- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Task

- Find all subgraph isomorphisms $\varphi : V(\mathcal{P}) \to V(\mathcal{T})$
- I.e., if
  1. $\varphi(a) = u$
  2. $\varphi(b) = v$
  3. $(a, b) \in E(\mathcal{P})$

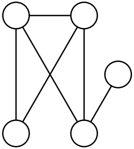
  then must have $(u, v) \in E(\mathcal{T})$

Subgraph Isomorphism Example

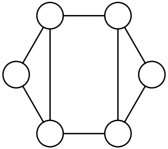


Pattern

Subgraph Isomorphism Example



Pattern



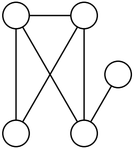
Target

Subgraph Isomorphism Example

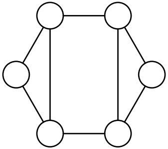
Pattern *Target*

No subgraph isomorphism

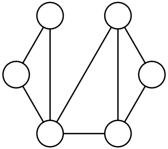
Subgraph Isomorphism Example



Pattern



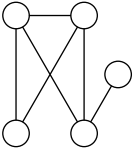
Target



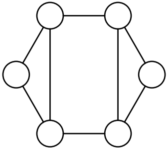
2nd target

No subgraph isomorphism

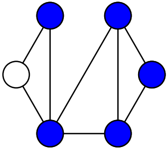
Subgraph Isomorphism Example



Pattern



Target

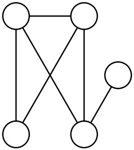


2nd target

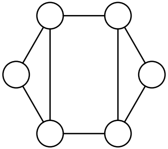
No subgraph isomorphism

Has subgraph isomorphism

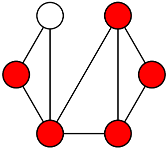
Subgraph Isomorphism Example



Pattern



Target



2nd target

No subgraph isomorphism

Has subgraph isomorphism
In fact, two of them

The Challenge

Subgraph isomorphism important in

- biochemistry
- compiler construction
- computer vision
- plagiarism and malware detection
- et cetera...

The Challenge

Subgraph isomorphism important in

- biochemistry
- compiler construction
- computer vision
- plagiarism and malware detection
- et cetera...

But computationally very challenging!

1. How to solve efficiently?

The Challenge

Subgraph isomorphism important in

- biochemistry
- compiler construction
- computer vision
- plagiarism and malware detection
- et cetera...

But computationally very challenging!

1. How to solve efficiently?
2. How do we know if answer is correct?

The Challenge

Subgraph isomorphism important in

- biochemistry
- compiler construction
- computer vision
- plagiarism and malware detection
- et cetera. . .

But computationally very challenging!

1. How to solve efficiently?
2. How do we know if answer is correct?
   (In particular, that we found **all** subgraph isomorphisms)

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm can be formalized in cutting planes proof system

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm can be formalized in cutting planes proof system
- Consequences:
  1. Produce efficient proofs of correctness with low overhead

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm can be formalized in cutting planes proof system
- Consequences:
  1. Produce efficient proofs of correctness with low overhead *(hopefully)*

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm can be formalized in cutting planes proof system
- Consequences:
  1. Produce efficient proofs of correctness with low overhead *(hopefully)*
  2. Learn pseudo-Boolean no-goods ⇒ exponential speed-up

This Work

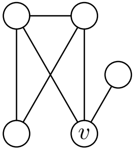
- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm can be formalized in cutting planes proof system
- Consequences:
  1. Produce efficient proofs of correctness with low overhead *(hopefully)*
  2. Learn pseudo-Boolean no-goods $\Rightarrow$ exponential speed-up *(maybe)*

Graph Notation and Terminology

- Undirected graphs $\mathcal{G}$ with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$
- No loops in this talk (for simplicity)
- Neighbours $N_{\mathcal{G}}(v) = \{u \mid (u,v) \in E(\mathcal{G})\}$
- Degree $\deg_{\mathcal{G}}(v) = \left|N_{\mathcal{G}}(v)\right|$
- Degree sequence $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$

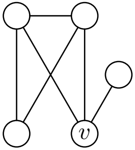
Graph Notation and Terminology

- Undirected graphs $\mathcal{G}$ with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$
- No loops in this talk (for simplicity)
- Neighbours $N_{\mathcal{G}}(v) = \{u \mid (u, v) \in E(\mathcal{G})\}$
- Degree $\deg_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$



Graph Notation and Terminology

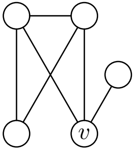
- Undirected graphs $\mathcal{G}$ with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$
- No loops in this talk (for simplicity)
- Neighbours $N_{\mathcal{G}}(v) = \{u \mid (u,v) \in E(\mathcal{G})\}$
- Degree $\deg_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$



$$\deg(v) = 3$$

Graph Notation and Terminology

- Undirected graphs $\mathcal{G}$ with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$
- No loops in this talk (for simplicity)
- Neighbours $N_{\mathcal{G}}(v) = \{u \mid (u,v) \in E(\mathcal{G})\}$
- Degree $\deg_{\mathcal{G}}(v) = \left|N_{\mathcal{G}}(v)\right|$
- Degree sequence $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$



$\deg(v) = 3$
$\mathrm{degseq}(v) = (3,3,1)$

Preprocessing Using Degree and Degree Sequence

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Preprocessing Using Degree and Degree Sequence

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Preprocessing

1. If $\left|V(\mathcal{P})\right| > \left|V(\mathcal{T})\right|$, then no solution

Preprocessing Using Degree and Degree Sequence

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Preprocessing

1. If $|V(\mathcal{P})| > |V(\mathcal{T})|$, then no solution
2. If $\deg_{\mathcal{P}}(a) > \deg_{\mathcal{T}}(u)$, then $a \not\mapsto u$

Preprocessing Using Degree and Degree Sequence

Input

- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

Preprocessing

1. If $|V(\mathcal{P})| > |V(\mathcal{T})|$, then no solution
2. If $\deg_{\mathcal{P}}(a) > \deg_{\mathcal{T}}(u)$, then $a \not\mapsto u$
3. If $\mathrm{degseq}_{\mathcal{P}}(a) \not\leq \mathrm{degseq}_{\mathcal{T}}(u)$ pointwise, then $a \not\mapsto u$

Preprocessing Using Shapes

Shapes

- Choose special shape graphs $\mathcal{S}$ with 2 special vertices s, t
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V\left(\mathcal{G}^{\mathcal{S}}\right) = V(\mathcal{G})$
  2. edges $(u, v) \in E\left(\mathcal{G}^{\mathcal{S}}\right)$ iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $s \mapsto u$ and $t \mapsto v$

Preprocessing Using Shapes

Shapes

- Choose special shape graphs $\mathcal{S}$ with 2 special vertices s, t
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V\big(\mathcal{G}^{\mathcal{S}}\big) = V(\mathcal{G})$
  2. edges $(u, v) \in E\big(\mathcal{G}^{\mathcal{S}}\big)$ iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $s \mapsto u$ and $t \mapsto v$

Further preprocessing

- If
  1. $a \mapsto u$
  2. $b \mapsto v$
  3. $(a, b) \in E\big(\mathcal{P}^{\mathcal{S}}\big)$

  then must have $(u, v) \in E\big(\mathcal{T}^{\mathcal{S}}\big)$

  (Since $\mathcal{S}$ "local subgraph" of $\mathcal{P}$, has to be "local subgraph" also of $\mathcal{T}$)

Preprocessing Using Shapes

Shapes

- Choose special shape graphs $\mathcal{S}$ with 2 special vertices s, t
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V\left(\mathcal{G}^{\mathcal{S}}\right) = V(\mathcal{G})$
  2. edges $(u, v) \in E\left(\mathcal{G}^{\mathcal{S}}\right)$ iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $s \mapsto u$ and $t \mapsto v$

Further preprocessing

- If
  1. $a \mapsto u$
  2. $b \mapsto v$
  3. $(a, b) \in E\left(\mathcal{P}^{\mathcal{S}}\right)$

  then must have $(u, v) \in E\left(\mathcal{T}^{\mathcal{S}}\right)$
  (Since $\mathcal{S}$ "local subgraph" of $\mathcal{P}$, has to be "local subgraph" also of $\mathcal{T}$)
- So repeat degree & degree sequence preprocessing for shaped graphs

Preprocessing Using Shapes

Shapes

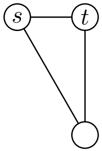
- Choose special shape graphs $\mathcal{S}$ with 2 special vertices s, t
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V\big(\mathcal{G}^{\mathcal{S}}\big) = V(\mathcal{G})$
  2. edges $(u, v) \in E\big(\mathcal{G}^{\mathcal{S}}\big)$ iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $s \mapsto u$ and $t \mapsto v$

Further preprocessing

- If
  1. $a \mapsto u$
  2. $b \mapsto v$
  3. $(a, b) \in E\big(\mathcal{P}^{\mathcal{S}}\big)$

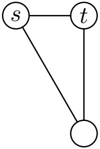
  then must have $(u, v) \in E\big(\mathcal{T}^{\mathcal{S}}\big)$
  (Since $\mathcal{S}$ "local subgraph" of $\mathcal{P}$, has to be "local subgraph" also of $\mathcal{T}$)
- So repeat degree & degree sequence preprocessing for shaped graphs
- Plus do some other stuff that we're skipping in this talk

Example of Preprocessing Using Shapes

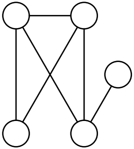


Shape

Example of Preprocessing Using Shapes

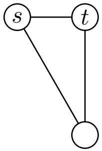


Shape

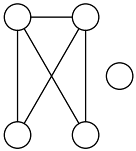


Pattern

Example of Preprocessing Using Shapes

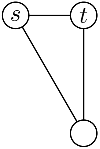


Shape

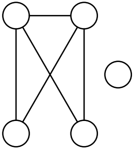


Pattern shaped

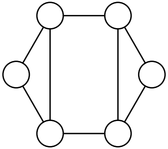
Example of Preprocessing Using Shapes



Shape

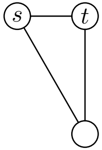


Pattern shaped

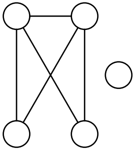


Target

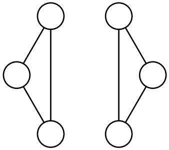
Example of Preprocessing Using Shapes



Shape

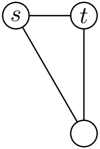


Pattern shaped

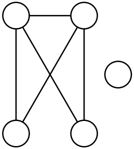


Target shaped

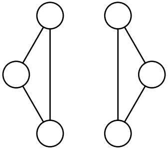
Example of Preprocessing Using Shapes



Shape



Pattern shaped



Target shaped

Now obvious that there can be no subgraph isomorphism!

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$
- Pick a with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$
- Pick a with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_{\mathcal{P}}(a)$ for assigned a to $D(b) \cap N_{\mathcal{T}}(u)$
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $\left|D(b)\right| = 1$

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$
- Pick a with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_{\mathcal{P}}(a)$ for assigned a to $D(b) \cap N_{\mathcal{T}}(u)$
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $\left|D(b)\right| = 1$
- Run all-different propagation
  If $\exists A$ with $D(A) = \bigcup_{a \in A} D(a)$ such that
  1. $|D(A)| < |A| \Rightarrow$ contradiction
  2. $|D(A)| = |A| \Rightarrow$ erase $D(A)$ from other domains

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$
- Pick a with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_{\mathcal{P}}(a)$ for assigned a to $D(b) \cap N_{\mathcal{T}}(u)$
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $\left|D(b)\right| = 1$
- Run all-different propagation
  If $\exists A$ with $D(A) = \bigcup_{a \in A} D(a)$ such that
  1. $|D(A)| < |A| \Rightarrow$ contradiction
  2. $|D(A)| = |A| \Rightarrow$ erase $D(A)$ from other domains
- Repeat from top of slide

Main Search Loop (Very Rough Outline)

- For every $a \in V(\mathcal{P})$ maintain possible domain $D(a) \subseteq V(\mathcal{T})$
- Pick a with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_{\mathcal{P}}(a)$ for assigned a to $D(b) \cap N_{\mathcal{T}}(u)$
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $\left|D(b)\right| = 1$
- Run all-different propagation
  If $\exists A$ with $D(A) = \bigcup_{a \in A} D(a)$ such that
  1. $|D(A)| < |A| \Rightarrow$ contradiction
  2. $|D(A)| = |A| \Rightarrow$ erase $D(A)$ from other domains
- Repeat from top of slide
- Backtrack at failure (or when solution found)

Pseudo-Boolean Constraints

In this talk, "pseudo-Boolean" (PB) refers to 0-1 integer linear constraints

Convenient to use non-negative linear combinations of literals, a.k.a. normalized form

$$\sum_i a_i \ell_i \geq A$$

- coefficients a_i: non-negative integers
- degree (of falsity) A: positive integer
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

Pseudo-Boolean Constraints

In this talk, "pseudo-Boolean" (PB) refers to 0-1 integer linear constraints

Convenient to use non-negative linear combinations of literals, a.k.a. normalized form

$$\sum_i a_i \ell_i \geq A$$

- coefficients a_i: non-negative integers
- degree (of falsity) A: positive integer
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

In what follows:

- all constraints assumed to be implicitly normalized
- "$\sum_i a_i \ell_i \leq A$" is syntactic sugar for "$\sum_i a_i \overline{\ell}_i \geq -A + \sum_i a_i$"
- "$=$" is syntactic sugar for two inequalities "$\geq$" and "$\leq$"

Examples of Pseudo-Boolean Constraints

1. Clauses are pseudo-Boolean constraints

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

(So can view CNF formula as collection of pseudo-Boolean constraints)

Examples of Pseudo-Boolean Constraints

1. Clauses are pseudo-Boolean constraints

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

(So can view CNF formula as collection of pseudo-Boolean constraints)

2. Cardinality constraints

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$$

Examples of Pseudo-Boolean Constraints

1. Clauses are pseudo-Boolean constraints

$$x \vee \overline{y} \vee z \quad \Leftrightarrow \quad x + \overline{y} + z \geq 1$$

(So can view CNF formula as collection of pseudo-Boolean constraints)

2. Cardinality constraints

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3$$

3. General constraints

$$x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7$$

Cutting Planes [CCT87]

Literal axioms $$\frac{}{\ell_i \geq 0}$$

Linear combination $$\frac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B \geq 0]$$

Division $$\frac{\sum_i a_i \ell_i \geq A}{\sum_i \lceil a_i / c \rceil \ell_i \geq \lceil A/c \rceil} \quad [c > 0]$$

More About Cutting Planes

A toy example:

$$\frac{6x + 2y + 3z \geq 5 \qquad x + 2y + w \geq 1}{(6x + 2y + 3z) + 2(x + 2y + w) \geq 5 + 2 \cdot 1} \text{ Linear combination}$$

More About Cutting Planes

A toy example:

$$\frac{6x + 2y + 3z \geq 5 \qquad x + 2y + w \geq 1}{8x + 6y + 3z + 2w \geq 7} \text{ Linear combination}$$

More About Cutting Planes

A toy example:

$$\dfrac{\dfrac{6x + 2y + 3z \geq 5 \qquad x + 2y + w \geq 1}{8x + 6y + 3z + 2w \geq 7}\ \text{Linear combination}}{3x + 2y + z + w \geq 3}\ \text{Division}$$

More About Cutting Planes

A toy example:

$$\dfrac{\dfrac{6x + 2y + 3z \geq 5 \qquad x + 2y + w \geq 1}{8x + 6y + 3z + 2w \geq 7}\ \text{Linear combination}}{3x + 2y + z + w \geq 3}\ \text{Division}$$

- Literal axioms and linear combinations sound also over the reals
- Division is where the power of cutting planes lies
- Exponentially stronger than resolution/CDCL [Hak85, CCT87]

Subgraph Isomorphism as a Pseudo-Boolean Formula

Recall:

- Pattern graph $\mathcal{P}$ with $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with $V(\mathcal{T}) = \{u, v, w, \ldots\}$
- No loops (for simplicity)

Subgraph Isomorphism as a Pseudo-Boolean Formula

Recall:

- Pattern graph $\mathcal{P}$ with $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with $V(\mathcal{T}) = \{u, v, w, \ldots\}$
- No loops (for simplicity)

Pseudo-Boolean encoding

$$\sum_{v \in V(\mathcal{T})} x_{a \mapsto v} = 1 \qquad \text{[every } a \text{ maps somewhere]}$$

$$\sum_{b \in V(\mathcal{P})} \overline{x}_{b \mapsto u} \geq \left|V(\mathcal{P})\right| - 1 \qquad \text{[mapping is one-to-one]}$$

$$\overline{x}_{a \mapsto u} + \sum_{v \in N(u)} x_{b \mapsto v} \geq 1 \qquad \text{[edge } (a, b) \text{ maps to edge } (u, v)\text{]}$$

Key Finding

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Key Finding

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Means that

1. Solver can justify each step by writing local formal derivation

Key Finding

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Means that

1. Solver can justify each step by writing local formal derivation
2. Local derivations can be concatenated to global proof of correctness

Key Finding

All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Means that

1. Solver can justify each step by writing local formal derivation
2. Local derivations can be concatenated to global proof of correctness
3. Proof checkable by stand-alone verifier
   - that knows nothing about graphs
   - in time comparable to the solver execution

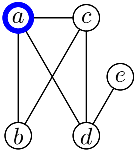
Key Finding

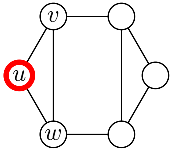
All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Means that

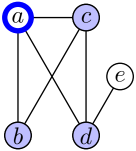
1. Solver can justify each step by writing local formal derivation
2. Local derivations can be concatenated to global proof of correctness
3. Proof checkable by stand-alone verifier
   - that knows nothing about graphs
   - ~~in time comparable to the solver execution~~
     in time hopefully not much larger than solver execution

Example: Degree Preprocessing with PB Reasoning

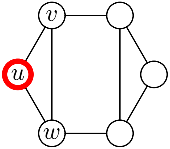




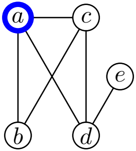
Example: Degree Preprocessing with PB Reasoning



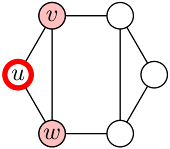
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$



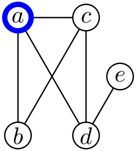
Example: Degree Preprocessing with PB Reasoning



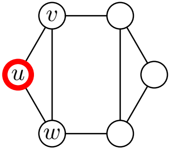
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} \geq 4$$
$$\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} \geq 4$$



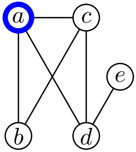
Example: Degree Preprocessing with PB Reasoning



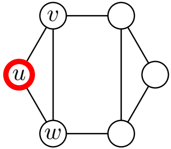
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} \geq 4$$
$$\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} \geq 4$$
$$x_{a\mapsto v} \geq 0$$
$$x_{a\mapsto w} \geq 0$$
$$x_{e\mapsto v} \geq 0$$
$$x_{e\mapsto w} \geq 0$$



Example: Degree Preprocessing with PB Reasoning

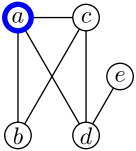


$$
\begin{aligned}
\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} &\geq 1\\
\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} &\geq 4\\
\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} &\geq 4\\
x_{a\mapsto v} &\geq 0\\
x_{a\mapsto w} &\geq 0\\
x_{e\mapsto v} &\geq 0\\
x_{e\mapsto w} &\geq 0
\end{aligned}
$$

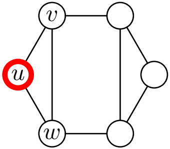


Sum up all constraints & divide by 3 to obtain

Example: Degree Preprocessing with PB Reasoning



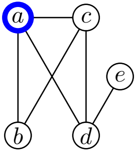
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} \geq 4$$
$$\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} \geq 4$$
$$x_{a\mapsto v} \geq 0$$
$$x_{a\mapsto w} \geq 0$$
$$x_{e\mapsto v} \geq 0$$
$$x_{e\mapsto w} \geq 0$$



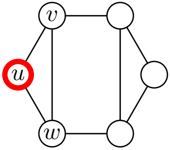
Sum up all constraints & divide by 3 to obtain

$$3\overline{x}_{a\mapsto u} + 10 \geq 11$$

Example: Degree Preprocessing with PB Reasoning



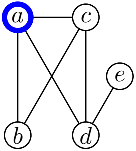
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} \geq 4$$
$$\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} \geq 4$$
$$x_{a\mapsto v} \geq 0$$
$$x_{a\mapsto w} \geq 0$$
$$x_{e\mapsto v} \geq 0$$
$$x_{e\mapsto w} \geq 0$$



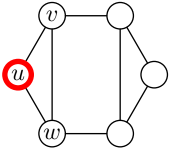
Sum up all constraints & divide by 3 to obtain

$$3\overline{x}_{a\mapsto u} \geq 1$$

Example: Degree Preprocessing with PB Reasoning



$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto v} + \overline{x}_{e\mapsto v} \geq 4$$
$$\overline{x}_{a\mapsto w} + \overline{x}_{b\mapsto w} + \overline{x}_{c\mapsto w} + \overline{x}_{d\mapsto w} + \overline{x}_{e\mapsto w} \geq 4$$
$$x_{a\mapsto v} \geq 0$$
$$x_{a\mapsto w} \geq 0$$
$$x_{e\mapsto v} \geq 0$$
$$x_{e\mapsto w} \geq 0$$



Sum up all constraints & divide by 3 to obtain

$$3\overline{x}_{a\mapsto u} \geq 1$$
$$\overline{x}_{a\mapsto u} \geq 1$$

Better Subgraph Solvers by Learning No-Goods?

- Subgraph isomorphism algorithm performs tree-like search
- Can we learn from failures and cut away larger parts of search space?

Better Subgraph Solvers by Learning No-Goods?

- Subgraph isomorphism algorithm performs tree-like search
- Can we learn from failures and cut away larger parts of search space?
- Has been tried using CDCL solvers — doesn't seem to work
- But CDCL only does resolution reasoning — very weak

Better Subgraph Solvers by Learning No-Goods?

- Subgraph isomorphism algorithm performs tree-like search
- Can we learn from failures and cut away larger parts of search space?
- Has been tried using CDCL solvers — doesn't seem to work
- But CDCL only does resolution reasoning — very weak
- Pseudo-Boolean solvers *Sat4j* [LP10] and *RoundingSat* [EN18] can be exponentially stronger
- E.g., can do all-different propagation, which CDCL can't

Better Subgraph Solvers by Learning No-Goods?

- Subgraph isomorphism algorithm performs tree-like search
- Can we learn from failures and cut away larger parts of search space?
- Has been tried using CDCL solvers — doesn't seem to work
- But CDCL only does resolution reasoning — very weak
- Pseudo-Boolean solvers *Sat4j* [LP10] and *RoundingSat* [EN18] can be exponentially stronger
- E.g., can do all-different propagation, which CDCL can't
- Remains to be seen whether this will fly in practice for subgraph isomorphism. . .

Take-Home Message

- Subgraph isomorphism important problem with many applications
- Can often be efficiently solved, but what about correctness?
- **This work:** Glasgow Subgraph Solver captured by cutting planes
- Consequences:
  1. Efficiently verifiable certificates of correctness
  2. Potential for exponential speed-up from pseudo-Boolean no-goods?
- **Caveat:** Still very much work in progress...
- **Question:** Can cutting planes formalize algorithms for other hard combinatorial problems in similar way?

Take-Home Message

- Subgraph isomorphism important problem with many applications
- Can often be efficiently solved, but what about correctness?
- **This work:** Glasgow Subgraph Solver captured by cutting planes
- Consequences:
  1. Efficiently verifiable certificates of correctness
  2. Potential for exponential speed-up from pseudo-Boolean no-goods?
- **Caveat:** Still very much work in progress. . .
- **Question:** Can cutting planes formalize algorithms for other hard combinatorial problems in similar way?

Thank you for your attention!

References I

[ADH$^+$19] Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. Sequential and parallel solution-biased search for subgraph algorithms. In *Proceedings of the 16th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR '19)*, June 2019. To appear.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[EN18] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, July 2018.

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[LP10] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[McC19] Ciaran McCreesh. Glasgow subgraph solver. `https://github.com/ciaranm/glasgow-subgraph-solver`, 2019.