Subgraph Isomorphism Meets Cutting Planes

Towards Verifiably Correct Constraint Programming

Jakob Nordström

University of Copenhagen and Lund University

Programming Languages and Theory of Computing Section
Department of Computer Science (DIKU)
January 14, 2020

Joint work with Stephan Gocht and 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

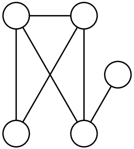
- Pattern graph $\mathcal{P}$ with vertices $V(\mathcal{P}) = \{a, b, c, \ldots\}$
- Target graph $\mathcal{T}$ with vertices $V(\mathcal{T}) = \{u, v, w, \ldots\}$

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. $(a, b) \in E(\mathcal{P})$

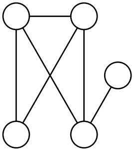
  then must have $(u, v) \in E(\mathcal{T})$

Subgraph Isomorphism Example

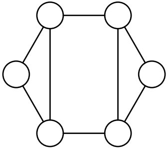


Pattern

Subgraph Isomorphism Example

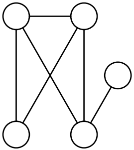


Pattern

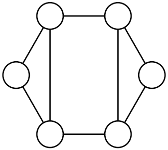


Target

Subgraph Isomorphism Example



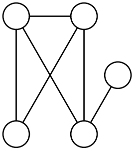
Pattern



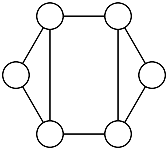
Target

No subgraph isomorphism

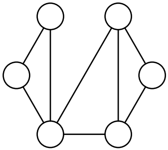
Subgraph Isomorphism Example



Pattern



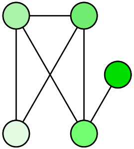
Target



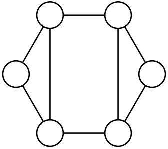
2nd target

No subgraph isomorphism

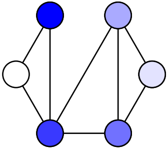
Subgraph Isomorphism Example



Pattern



Target

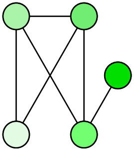


2nd target

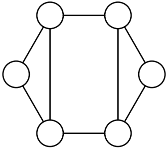
No subgraph isomorphism

Has subgraph isomorphism

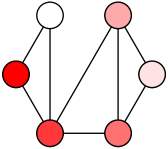
Subgraph Isomorphism Example



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. Even more importantly: How do we know 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. Even more importantly: How do we know 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 formalizable in cutting planes proof system

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm formalizable in cutting planes proof system
- As a consequence, can produce proofs of correctness
  1. with not too large overhead for solver
  2. efficiently verifiable by stand-alone proof checker

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm formalizable in cutting planes proof system
- As a consequence, can produce proofs of correctness
  1. with not too large overhead for solver
  2. efficiently verifiable by stand-alone proof checker
- Results extend also to other state-of-the-art subgraph solvers

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm formalizable in cutting planes proof system
- As a consequence, can produce proofs of correctness
  1. with not too large overhead for solver
  2. efficiently verifiable by stand-alone proof checker
- Results extend also to other state-of-the-art subgraph solvers
- And to constraint programming in general... [EGMN20]

This Work

- Analyze Glasgow Subgraph Solver [ADH+19, McC19]
- Show algorithm formalizable in cutting planes proof system
- As a consequence, can produce proofs of correctness
  1. with not too large overhead for solver
  2. efficiently verifiable by stand-alone proof checker
- Results extend also to other state-of-the-art subgraph solvers
- And to constraint programming in general... [EGMN20]
- Intriguing possibility: learn pseudo-Boolean no-goods $\Rightarrow$ exponential speed-ups!?

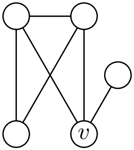
Outline

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
  $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$

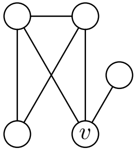
Graph Notation and Terminology

- 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
  $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$



Graph Notation and Terminology

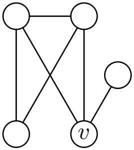
- Undirected graphs $\mathcal{G}$ with vertices $V(\mathcal{G})$ and edges $E(\mathcal{G})$
- No loops in this talk (for simplicity)
- Neighbours $N_{\mathcal{G}}(v) = \{u \mid (u, v) \in E(\mathcal{G})\}$
- Degree $\deg_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence
  $\mathrm{degseq}_{\mathcal{G}}(v)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



$\deg(v) = 3$

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 $\deg_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence
  $\mathrm{degseq}_{\mathcal{G}}(v) = sort_{>}(\{\deg_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\})$



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

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 $|V(\mathcal{P})| > |V(\mathcal{T})|$, 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 $\left|V(\mathcal{P})\right| > \left|V(\mathcal{T})\right|$, 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 shape graph $\mathcal{S}$ with 2 special vertices σ, τ
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V(\mathcal{G})$
  2. edges (u, v) iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $\sigma \mapsto u$ & $\tau \mapsto v$

Preprocessing Using Shapes

Shapes

- Choose shape graph $\mathcal{S}$ with 2 special vertices σ, τ
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V(\mathcal{G})$
  2. edges (u, v) iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $\sigma \mapsto u$ & $\tau \mapsto v$

Further preprocessing

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

  then must have $(u, v) \in E(\mathcal{T}^{\mathcal{S}})$
  ($\mathcal{S}$ "local subgraph" of $\mathcal{P} \Rightarrow$ "local subgraph" also of $\mathcal{T}$)

Preprocessing Using Shapes

Shapes

- Choose shape graph $\mathcal{S}$ with 2 special vertices σ, τ
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V(\mathcal{G})$
  2. edges (u, v) iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $\sigma \mapsto u$ & $\tau \mapsto v$

Further preprocessing

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

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

Preprocessing Using Shapes

Shapes

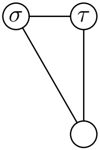
- Choose shape graph $\mathcal{S}$ with 2 special vertices σ, τ
- Shaped graph $\mathcal{G}^{\mathcal{S}}$ has
  1. vertices $V(\mathcal{G})$
  2. edges (u, v) iff $\mathcal{S}$ subgraph of $\mathcal{G}$ with $\sigma \mapsto u$ & $\tau \mapsto v$

Further preprocessing

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

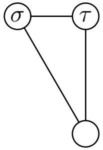
  then must have $(u, v) \in E(\mathcal{T}^{\mathcal{S}})$
  ($\mathcal{S}$ "local subgraph" of $\mathcal{P} \Rightarrow$ "local subgraph" also of $\mathcal{T}$)
- So repeat degree & degree sequence preprocessing for shaped graphs
- Plus do some other stuff that we're skipping in this talk. . .

Example of Preprocessing Using Shapes

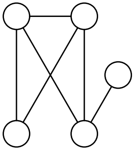


Shape

Example of Preprocessing Using Shapes

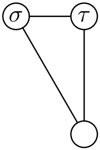


Shape

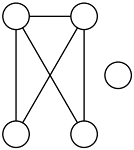


Pattern

Example of Preprocessing Using Shapes

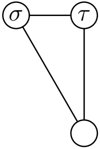


Shape

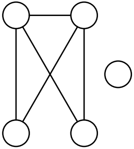


Pattern shaped

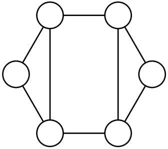
Example of Preprocessing Using Shapes



Shape

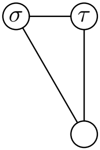


Pattern shaped

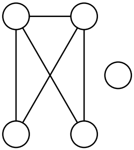


Target

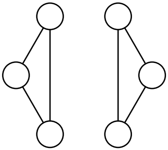
Example of Preprocessing Using Shapes



Shape

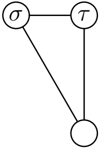


Pattern shaped

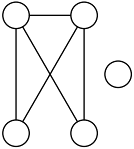


Target shaped

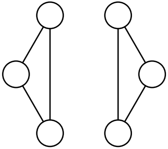
Example of Preprocessing Using Shapes



Shape



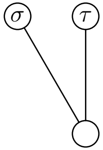
Pattern shaped



Target shaped

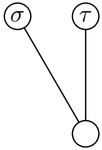
Now obvious that there can be no subgraph isomorphism!

Second Example of Preprocessing Using Shapes

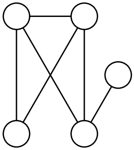


Shape

Second Example of Preprocessing Using Shapes

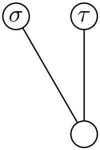


Shape

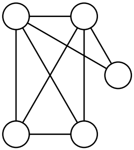


Pattern

Second Example of Preprocessing Using Shapes

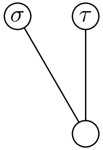


Shape

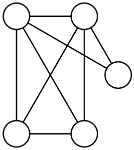


Pattern shaped

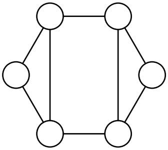
Second Example of Preprocessing Using Shapes



Shape

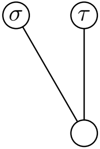


Pattern shaped

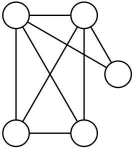


Target

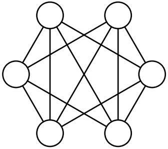
Second Example of Preprocessing Using Shapes



Shape

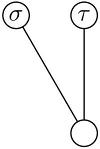


Pattern shaped

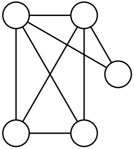


Target shaped

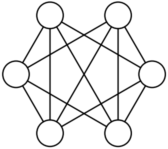
Second Example of Preprocessing Using Shapes



Shape



Pattern shaped



Target shaped

Maybe not as obviously enlightening. . .

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 $D(b) \cap N_{\mathcal{T}}(u)$ (do this also for shaped graphs)
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $|D(b)| = 1$

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 $D(b) \cap N_{\mathcal{T}}(u)$ (do this also for shaped graphs)
  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 $D(b) \cap N_{\mathcal{T}}(u)$ (do this also for shaped graphs)
  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 $D(b) \cap N_{\mathcal{T}}(u)$ (do this also for shaped graphs)
  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 chained into global correctness proof

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 chained into global correctness proof
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 chained into global correctness proof
3. Proof checkable by stand-alone verifier
    - that knows nothing about graphs
    - ~~in time comparable to the solver execution~~
      in time hopefully not too much larger than solver execution (work in progress on optimizing this)

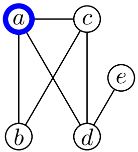
Key Finding

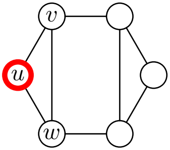
All reasoning steps in Glasgow Subgraph Solver can be formalized efficiently in the cutting planes proof system

Means that

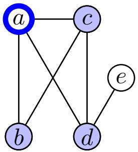
1. Solver can justify each step by writing local formal derivation
2. Local derivations can be chained into global correctness proof
3. Proof checkable by stand-alone verifier
    - that knows nothing about graphs
    - ~~in time comparable to the solver execution~~
      in time hopefully not too much larger than solver execution (work in progress on optimizing this)
4. Further interesting features:
    - Even for buggy solver, a correct proof is always accepted
    - Even for formally verified solver that gets whacked by cosmic radiation/hardware failure, wrong proof will always be rejected

Example: Degree Preprocessing with PB Reasoning





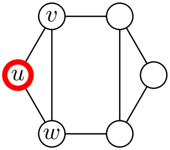
Example: Degree Preprocessing with PB Reasoning



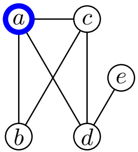
$$\overline{x}_{a \mapsto u} + x_{b \mapsto v} + x_{b \mapsto w} \geq 1$$

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

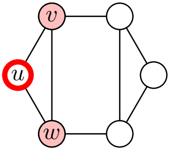
$$\overline{x}_{a \mapsto u} + x_{d \mapsto v} + x_{d \mapsto w} \geq 1$$



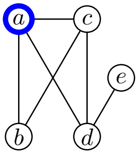
Example: Degree Preprocessing with PB Reasoning



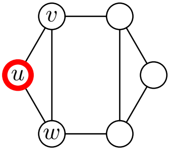
$$\overline{x}_{a \mapsto u} + x_{b \mapsto v} + x_{b \mapsto w} \geq 1$$
$$\overline{x}_{a \mapsto u} + x_{c \mapsto v} + x_{c \mapsto w} \geq 1$$
$$\overline{x}_{a \mapsto u} + x_{d \mapsto v} + x_{d \mapsto w} \geq 1$$
$$\overline{x}_{a \mapsto v} + \overline{x}_{b \mapsto v} + \overline{x}_{c \mapsto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



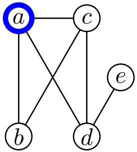
Example: Degree Preprocessing with PB Reasoning



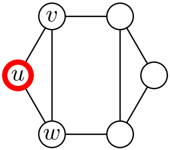
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto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



Example: Degree Preprocessing with PB Reasoning

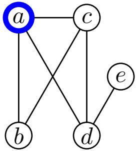


$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto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

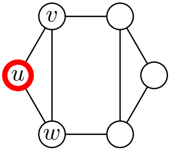


Sum up all constraints & divide by 3 to obtain

Example: Degree Preprocessing with PB Reasoning



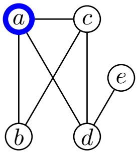
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto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



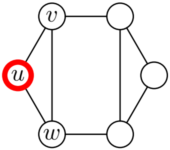
Sum up all constraints & divide by 3 to obtain

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

Example: Degree Preprocessing with PB Reasoning



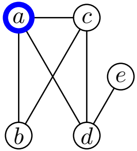
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto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



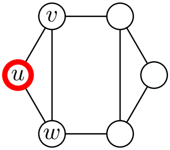
Sum up all constraints & divide by 3 to obtain

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

Example: Degree Preprocessing with PB Reasoning



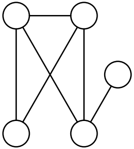
$$\overline{x}_{a\mapsto u} + x_{b\mapsto v} + x_{b\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{c\mapsto v} + x_{c\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto u} + x_{d\mapsto v} + x_{d\mapsto w} \geq 1$$
$$\overline{x}_{a\mapsto v} + \overline{x}_{b\mapsto v} + \overline{x}_{c\mapsto v} + \overline{x}_{d\mapsto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



Sum up all constraints & divide by 3 to obtain

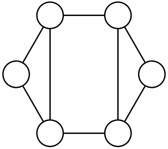
$$3\overline{x}_{a\mapsto u} \geq 1$$
$$\overline{x}_{a\mapsto u} \geq 1$$

Graph Input Format



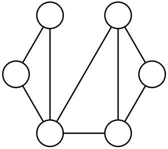
Pattern

```
5
3 1 3 4
3 0 3 4
1 3
3 0 1 2
2 0 1
```



Target 1

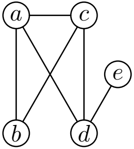
```
6
3 1 4 5
3 0 2 3
3 1 3
3 1 2 4
3 0 3 5
2 0 4
```



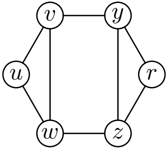
Target 2

```
6
2 4 5
3 2 3 4
2 1 3
3 1 2 4
4 0 1 3 5
2 0 4
```

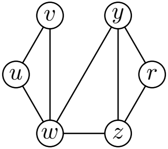
Graph Input Format



Pattern



Target 1



Target 2

Pattern	Target 1	Target 2
	`v,y`	`v,w`
`a,b`	`v,w`	`u,v`
`a,c`	`u,v`	`y,r`
`a,d`	`y,r`	`y,z`
`b,c`	`y,z`	`y,w`
`c,d`	`r,z`	`r,z`
`d,e`	`z,w`	`z,w`
	`u,w`	`u,w`

Pseudo-Boolean Encoding for Mapping Pattern to Target 1

```
* #variable= 30 #constraint= 88
* pattern vertex domain constraints
1 a_v 1 a_y 1 a_w 1 a_u 1 a_r 1 a_z >= 1 ;
-1 a_v -1 a_y -1 a_w -1 a_u -1 a_r -1 a_z >= -1 ;
1 c_v 1 c_y 1 c_w 1 c_u 1 c_r 1 c_z >= 1 ;
-1 c_v -1 c_y -1 c_w -1 c_u -1 c_r -1 c_z >= -1 ;
1 d_v 1 d_y 1 d_w 1 d_u 1 d_r 1 d_z >= 1 ;
-1 d_v -1 d_y -1 d_w -1 d_u -1 d_r -1 d_z >= -1 ;
1 b_v 1 b_y 1 b_w 1 b_u 1 b_r 1 b_z >= 1 ;
-1 b_v -1 b_y -1 b_w -1 b_u -1 b_r -1 b_z >= -1 ;
1 e_v 1 e_y 1 e_w 1 e_u 1 e_r 1 e_z >= 1 ;
-1 e_v -1 e_y -1 e_w -1 e_u -1 e_r -1 e_z >= -1 ;
* injectivity constraint for target vertices
-1 a_v -1 c_v -1 d_v -1 b_v -1 e_v >= -1 ;
-1 a_y -1 c_y -1 d_y -1 b_y -1 e_y >= -1 ;
-1 a_w -1 c_w -1 d_w -1 b_w -1 e_w >= -1 ;
-1 a_u -1 c_u -1 d_u -1 b_u -1 e_u >= -1 ;
-1 a_r -1 c_r -1 d_r -1 b_r -1 e_r >= -1 ;
-1 a_z -1 c_z -1 d_z -1 b_z -1 e_z >= -1 ;
* adjacency for edge a -- c mapping a to v
1 ~a_v 1 c_y 1 c_w 1 c_u >= 1 ;
* adjacency for edge a -- d mapping a to v
1 ~a_v 1 d_y 1 d_w 1 d_u >= 1 ;
* adjacency for edge a -- b mapping a to v
1 ~a_v 1 b_y 1 b_w 1 b_u >= 1 ;
* adjacency for edge a -- c mapping a to y
1 ~a_y 1 c_v 1 c_r 1 c_z >= 1 ;
* adjacency for edge a -- d mapping a to y
1 ~a_y 1 d_v 1 d_r 1 d_z >= 1 ;
* adjacency for edge a -- b mapping a to y
1 ~a_y 1 b_v 1 b_r 1 b_z >= 1 ;
* adjacency for edge a -- c mapping a to w
1 ~a_w 1 c_v 1 c_u 1 c_z >= 1 ;
* adjacency for edge a -- d mapping a to w
1 ~a_w 1 d_v 1 d_u 1 d_z >= 1 ;
* adjacency for edge a -- b mapping a to w
1 ~a_w 1 b_v 1 b_u 1 b_z >= 1 ;
* adjacency for edge a -- c mapping a to u
1 ~a_u 1 c_v 1 c_w >= 1 ;
* adjacency for edge a -- d mapping a to u
1 ~a_u 1 d_v 1 d_w >= 1 ;
* adjacency for edge a -- b mapping a to u
1 ~a_u 1 b_v 1 b_w >= 1 ;
* adjacency for edge a -- c mapping a to r
1 ~a_r 1 c_y 1 c_z >= 1 ;
* adjacency for edge a -- d mapping a to r
1 ~a_r 1 d_y 1 d_z >= 1 ;
* adjacency for edge a -- b mapping a to r
1 ~a_r 1 b_y 1 b_z >= 1 ;
* adjacency for edge a -- c mapping a to z
1 ~a_z 1 c_y 1 c_w 1 c_r >= 1 ;
* adjacency for edge a -- d mapping a to z
1 ~a_z 1 d_y 1 d_w 1 d_r >= 1 ;
* adjacency for edge a -- b mapping a to z
1 ~a_z 1 b_y 1 b_w 1 b_r >= 1 ;
* adjacency for edge c -- a mapping c to v
1 ~c_v 1 a_y 1 a_w 1 a_u >= 1 ;
. . .
```

Proof Logging Format and Rules (Excerpt)

Formulas: Extension of *OPB* (`www.cril.univ-artois.fr/PB12/format.pdf`)
Proofs: *VeriPB* (`github.com/StephanGocht/VeriPB`, [EGMN20])

Every constraint gets line number, which can be used to refer to the constraint

- `f [nProblemConstraints] 0`
  Load input formula from (specified) file
- `p [sequence of operations in reverse polish notation] 0`
  Derive constraint by addition, scalar multiplication and division
- `u [PB constraint]`
  Add PB constraint as valid if negation unit propagates to contradiction
- `j [constraintId] [PB constraint]`
  Add PB constraint as valid if implied by constraint on given line
- `d [constraintId1] [constraintId2] [constraintId3] ... 0`
  Delete constraints from database of derived PB constraints
- `v [literal] [literal] ... 0`
  Check that partial assignment propagates to solution; add the disjunction of the negations of these literals to mark solution as found
- `c [ConstraintId] 0`
  Verify that constraint on line `ConstraintId` *is* $0 \geq A$ *for some positive* A

Proof of No Subgraph Isomorphism for Pattern & Target 1

```
pseudo-Boolean proof version 1.0
f 88 0
* cannot map a to u due to degrees in graph pairs 0
p 26 27 + 28 + 11 + 13 + 0
j 89 1 ~xa_u >= 1 ;
d 89 0
* cannot map a to r due to degrees in graph pairs 0
p 29 30 + 31 + 12 + 16 + 0
j 91 1 ~xa_r >= 1 ;
d 91 0
* cannot map c to u due to degrees in graph pairs 0
p 56 57 + 58 + 11 + 13 + 0
j 93 1 ~xc_u >= 1 ;
d 93 0
* cannot map c to r due to degrees in graph pairs 0
p 59 60 + 61 + 12 + 16 + 0
j 95 1 ~xc_r >= 1 ;
d 95 0
* cannot map d to u due to degrees in graph pairs 0
p 74 75 + 76 + 11 + 13 + 0
j 97 1 ~xd_u >= 1 ;
d 97 0
* cannot map d to r due to degrees in graph pairs 0
p 77 78 + 79 + 12 + 16 + 0
j 99 1 ~xd_r >= 1 ;
d 99 0
* [0] guessing a=z
* unit propagating a=z
* hall set or violator size 3/3
p 1 5 + 7 + 12 + 13 + 16 + 0
* hall set or violator size 4/4
p 1 5 + 7 + 3 + 12 + 13 + 15 + 16 + 0
* unit propagating b=r
* hall set or violator size 3/3
p 1 3 + 5 + 12 + 15 + 16 + 0
* unit propagating c=y
* [1] propagation failure on a=z
u  1 ~xa_z >= 1 ;
* [0] guessing a=v
* unit propagating a=v
* hall set or violator size 3/3
p 1 5 + 7 + 11 + 12 + 13 + 0
* hall set or violator size 4/4
p 1 5 + 7 + 3 + 11 + 12 + 13 + 14 + 0
* unit propagating b=u
* hall set or violator size 3/3
p 1 3 + 5 + 11 + 13 + 14 + 0
* unit propagating c=w
* [1] propagation failure on a=v
u  1 ~xa_v >= 1 ;
* [0] guessing a=w
* unit propagating a=w
* hall set or violator size 3/3
p 1 5 + 7 + 11 + 13 + 16 + 0
* hall set or violator size 4/4
p 1 5 + 7 + 3 + 11 + 13 + 14 + 16 + 0
* unit propagating b=u
* hall set or violator size 3/3
p 1 3 + 5 + 11 + 13 + 14 + 0
* unit propagating c=v
* [1] propagation failure on a=w
u  1 ~xa_w >= 1 ;
* [0] guessing a=y
* unit propagating a=y
* hall set or violator size 3/3
p 1 5 + 7 + 11 + 12 + 16 + 0
* hall set or violator size 4/4
p 1 5 + 7 + 3 + 11 + 12 + 15 + 16 + 0
* unit propagating b=r
* hall set or violator size 3/3
p 1 3 + 5 + 12 + 15 + 16 + 0
* unit propagating c=z
* [1] propagation failure on a=y
u  1 ~xa_y >= 1 ;
* asserting that we've proved unsat
u >= 1 ;
c 117 0
```

Proof for Subgraph Isomorphisms for Pattern & Target 2

```
pseudo-Boolean proof version 1.0
f 88 0
* cannot map a to v due to degrees in graph pairs 0
p 17 18 + 19 + 12 + 13 + 0
j 89 1 ~xa_v >= 1 ;
d 89 0
* cannot map a to u due to degrees in graph pairs 0
p 23 24 + 25 + 11 + 12 + 0
j 91 1 ~xa_u >= 1 ;
d 91 0
* cannot map a to r due to degrees in graph pairs 0
p 29 30 + 31 + 14 + 16 + 0
j 93 1 ~xa_r >= 1 ;
d 93 0

 .  .  .

* [1] guessing e=u
* unit propagating e=u
# 3
* found solution a=z b=r c=y d=w e=u
v xa_z xb_r xc_y xd_w xe_u
# 2
* [2] backtracking
u  1 ~xa_z 1 ~xe_u >= 1 ;
w 3
* [1] guessing e=v
* unit propagating e=v
# 3
* found solution a=z b=r c=y d=w e=v
v xa_z xb_r xc_y xd_w xe_v
# 2
* [2] backtracking
u  1 ~xa_z 1 ~xe_v >= 1 ;
w 3
# 1
* [1] incorrect guess
u  1 ~xa_z >= 1 ;
w 2
* [0] guessing a=w
* unit propagating a=w
* hall set or violator size 3/3
p 1 3 + 5 + 12 + 14 + 16 + 0
* [1] propagation failure on a=w
u  1 ~xa_w >= 1 ;
* [0] guessing a=y
* unit propagating a=y
* hall set or violator size 3/3
p 1 5 + 7 + 12 + 14 + 16 + 0
* hall set or violator size 4/4
p 1 5 + 7 + 3 + 12 + 14 + 15 + 16 + 0
* unit propagating b=r
* hall set or violator size 3/3
p 1 3 + 5 + 14 + 15 + 16 + 0
* unit propagating c=z
* unit propagating d=w
# 2
* [1] guessing e=v
* unit propagating e=v
# 3
* found solution a=y b=r c=z d=w e=v
v xa_y xb_r xc_z xd_w xe_v
# 2
* [2] backtracking
u  1 ~xa_y 1 ~xe_v >= 1 ;
w 3
* [1] guessing e=u
* unit propagating e=u
# 3
* found solution a=y b=r c=z d=w e=u
v xa_y xb_r xc_z xd_w xe_u
# 2
* [2] backtracking
u  1 ~xa_y 1 ~xe_u >= 1 ;
w 3
# 1
* [1] incorrect guess
u  1 ~xa_y >= 1 ;
w 2
* asserting that we've proved unsat
u >= 1 ;
c 129 0
```

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 pseudo-Boolean reasoning using cutting planes
- Consequences:
  1. Efficiently verifiable certificates of correctness
  2. Potential for exponential speed-up from PB no-goods?
- **Caveat:** Further optimizations still needed...
- **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 pseudo-Boolean reasoning using cutting planes
- Consequences:
  1. Efficiently verifiable certificates of correctness
  2. Potential for exponential speed-up from PB no-goods?
- **Caveat:** Further optimizations still needed. . .
- **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.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, February 2020. To appear.

[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.

References II

[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, July 2010.

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