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** \( \varphi : V(\mathcal{P}) \rightarrow V(\mathcal{T}) \)
- 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

No subgraph isomorphism

Has subgraph isomorphism
Subgraph Isomorphism Example

**Pattern**

No subgraph isomorphism

**Target**

Has subgraph isomorphism
In fact, two of them

**2nd target**
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\textsuperscript{+}19, McC19]
This Work

- Analyze Glasgow Subgraph Solver [ADH\textsuperscript{+}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:
  - 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 $\Rightarrow$ exponential speed-up
This Work

- Analyze *Glasgow Subgraph Solver* \([\text{ADH}^+19, \text{McC}19]\)
- 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 (*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 $\text{deg}_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence $\text{degseq}_{\mathcal{G}}(v) = \text{sort}\cdot\{\text{deg}_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\}$
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 $N_G(v) = \{u \mid (u, v) \in E(\mathcal{G})\}$
- Degree $\deg_G(v) = |N_G(v)|$
- Degree sequence $\degseq_G(v) = sort_\succ(\{\deg_G(u) \mid u \in N_G(v)\})$
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 $\text{deg}_{\mathcal{G}}(v) = |N_{\mathcal{G}}(v)|$
- Degree sequence $\text{degseq}_{\mathcal{G}}(v) = \text{sort}>\left(\{\text{deg}_{\mathcal{G}}(u) \mid u \in N_{\mathcal{G}}(v)\}\right)$

$\text{deg}(v) = 3$
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 \( \degseq_\mathcal{G}(v) = \text{sort}\>(\{\deg_\mathcal{G}(u) \mid u \in N_\mathcal{G}(v)\}) \)

\[ \begin{align*}
\text{deg}(v) &= 3 \\
\text{degseq}(v) &= (3, 3, 1)
\end{align*} \]
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 $|V(\mathcal{P})| > |V(\mathcal{T})|$, then no solution
2. If $\deg_{\mathcal{P}}(a) > \deg_{\mathcal{T}}(u)$, then $a \not\rightarrow u$
3. If $\degseq_{\mathcal{P}}(a) \not\sim \degseq_{\mathcal{T}}(u)$ pointwise, then $a \not\leftrightarrow u$
Preprocessing Using Shapes

Shapes

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

Shapes

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

Further preprocessing

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

then must have $(u, v) \in E(T^S)$
(Since $S$ “local subgraph” of $P$, has to be “local subgraph” also of $T$)
Preprocessing Using Shapes

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

**Further preprocessing**
- If
  1. $a \mapsto u$
  2. $b \mapsto v$
  3. $(a, b) \in E(P^S)$
  then must have $(u, v) \in E(T^S)$
  (Since $S$ “local subgraph” of $P$, has to be “local subgraph” also of $T$)
- So repeat degree & degree sequence preprocessing for shaped graphs
Preprocessing Using Shapes

Shapes

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

Further preprocessing

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

  then must have $(u, v) \in E(T^S)$
  (Since $S$ “local subgraph” of $P$, has to be “local subgraph” also of $T$)

- 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

Now obvious that there can be no subgraph isomorphism!
Example of Preprocessing Using Shapes

Shape

Pattern

Now obvious that there can be no subgraph isomorphism!
Example of Preprocessing Using Shapes

Shape

Pattern shaped

Now obvious that there can be no subgraph isomorphism!
Example of Preprocessing Using Shapes

Shape

Pattern shaped

Target

Now obvious that there can be no subgraph isomorphism!
Example of Preprocessing Using Shapes

Shape

Pattern shaped

Target shaped

Now obvious that there can be no subgraph isomorphism!
Example of Preprocessing Using Shapes

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 \( a \in V(P) \) maintain possible domain \( D(a) \subseteq V(T) \)
- 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 \( |D(b)| = 1 \)
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 $D(b) \cap N_\mathcal{T}(u)$
  2. Propagate assignment for $b \in V(\mathcal{P})$ with $|D(b)| = 1$
- Run all-different propagation
  If $\exists A$ with $D(A) = \bigcup_{a \in A} D(a)$ such that
    1. $|D(A)| < |A|$ ⇒ contradiction
    2. $|D(A)| = |A|$ ⇒ erase $D(A)$ from other domains
Main Search Loop (Very Rough Outline)

- For every $a \in V(P)$ maintain possible domain $D(a) \subseteq V(T)$
- Pick $a$ with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_P(a)$ for assigned $a$ to $D(b) \cap N_T(u)$
  2. Propagate assignment for $b \in V(P)$ with $|D(b)| = 1$
- 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 $a \in V(P)$ maintain possible domain $D(a) \subseteq V(T)$
- Pick $a$ with smallest domain & iterate over $a \mapsto u$ for $u \in D(a)$
- Repeat until saturation
  1. Shrink domains of $b \in N_P(a)$ for assigned $a$ to $D(b) \cap N_T(u)$
  2. Propagate assignment for $b \in V(P)$ with $|D(b)| = 1$
- 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)
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 $\ell_i$: $x_i$ 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 \( \ell_i \): \( x_i \) 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

   \[ x \lor \overline{y} \lor z \iff x + \overline{y} + z \geq 1 \]

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

1. **Clauses** are pseudo-Boolean constraints

   \[ x \lor y \lor z \iff x + \overline{y} + z \geq 1 \]

   (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

   \[ x \lor \overline{y} \lor z \iff x + \overline{y} + z \geq 1 \]

   (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**

   \[ x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 \geq 7 \]
Cutting Planes [CCT87]

**Literal axioms**
\[ \ell_i \geq 0 \]

**Linear combination**
\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B \]
\[ [c_A, c_B \geq 0] \]

**Division**
\[ \frac{\sum_i a_i \ell_i \geq A}{\sum_i \lceil a_i / c \rceil \ell_i \geq \lceil A / c \rceil} \]
\[ [c > 0] \]
More About Cutting Planes

A toy example:

\[
\begin{align*}
6x + 2y + 3z & \geq 5 \\
\frac{(6x + 2y + 3z) + 2(x + 2y + w)}{x + 2y + w} & \geq 1 \\
\end{align*}
\]

Linear combination

Division is where the power of cutting planes lies

Literal axioms and linear combinations sound also over the reals

Exponentially stronger than resolution/CDCL [Hak85, CCT87]
More About Cutting Planes

A toy example:

\[
\begin{align*}
6x + 2y + 3z & \geq 5 \\
x + 2y + w & \geq 1
\end{align*}
\]

\[
\begin{align*}
8x + 6y + 3z + 2w & \geq 7
\end{align*}
\]

Linear combination
More About Cutting Planes

A toy example:

\[
\begin{align*}
6x + 2y + 3z & \geq 5 \\
\hline
x + 2y + w & \geq 1 \\
8x + 6y + 3z + 2w & \geq 7 \\
\hline
3x + 2y + z + w & \geq 3
\end{align*}
\]

- Linear combination
- 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].
More About Cutting Planes

A toy example:

\[
\begin{align*}
6x + 2y + 3z & \geq 5 \\
x + 2y + w & \geq 1
\end{align*}
\]

\[
\frac{8x + 6y + 3z + 2w}{3x + 2y + z + w} \geq 3
\]

- 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**

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

\[
\sum_{b \in V(\mathcal{P})} \overline{x}_{b \rightarrow u} \geq |V(\mathcal{P})| - 1 \quad [\text{mapping is one-to-one}]
\]

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

\[ \begin{align*} a & \quad c & \quad e \\ b & \quad d & \quad e \end{align*} \]

\[ \begin{align*} v & \quad u & \quad w \end{align*} \]

Sum up all constraints & divide by 3 to obtain.
Example: Degree Preprocessing with PB Reasoning

\[
\overline{x}_{a\rightarrow u} + x_{b\rightarrow v} + x_{b\rightarrow w} \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{c\rightarrow v} + x_{c\rightarrow w} \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{d\rightarrow v} + x_{d\rightarrow w} \geq 1
\]
Example: Degree Preprocessing with PB Reasoning

\[ \overline{x_{a \rightarrow u}} + x_{b \rightarrow v} + x_{b \rightarrow w} \geq 1 \]
\[ \overline{x_{a \rightarrow u}} + x_{c \rightarrow v} + x_{c \rightarrow w} \geq 1 \]
\[ \overline{x_{a \rightarrow u}} + x_{d \rightarrow v} + x_{d \rightarrow w} \geq 1 \]
\[ \overline{x_{a \rightarrow u}} + \overline{x_{b \rightarrow v}} + \overline{x_{c \rightarrow v}} + \overline{x_{d \rightarrow v}} + \overline{x_{e \rightarrow v}} \geq 4 \]
\[ \overline{x_{a \rightarrow w}} + \overline{x_{b \rightarrow w}} + \overline{x_{c \rightarrow w}} + \overline{x_{d \rightarrow w}} + \overline{x_{e \rightarrow w}} \geq 4 \]
Example: Degree Preprocessing with PB Reasoning

\[
\begin{align*}
\overline{x}_{a \rightarrow u} + x_{b \rightarrow v} + x_{b \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow u} + x_{c \rightarrow v} + x_{c \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow u} + x_{d \rightarrow v} + x_{d \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow v} + \overline{x}_{b \rightarrow v} + \overline{x}_{c \rightarrow v} + \overline{x}_{d \rightarrow v} + \overline{x}_{e \rightarrow v} & \geq 4 \\
\overline{x}_{a \rightarrow w} + \overline{x}_{b \rightarrow w} + \overline{x}_{c \rightarrow w} + \overline{x}_{d \rightarrow w} + \overline{x}_{e \rightarrow w} & \geq 4 \\
x_{a \rightarrow v} & \geq 0 \\
x_{a \rightarrow w} & \geq 0 \\
x_{e \rightarrow v} & \geq 0 \\
x_{e \rightarrow w} & \geq 0
\end{align*}
\]
Example: Degree Preprocessing with PB Reasoning

\[
\begin{align*}
\overline{x}_{a\rightarrow u} + x_{b\rightarrow v} + x_{b\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{c\rightarrow v} + x_{c\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{d\rightarrow v} + x_{d\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow v} + \overline{x}_{b\rightarrow v} + \overline{x}_{c\rightarrow v} + \overline{x}_{d\rightarrow v} + \overline{x}_{e\rightarrow v} & \geq 4 \\
\overline{x}_{a\rightarrow w} + \overline{x}_{b\rightarrow w} + \overline{x}_{c\rightarrow w} + \overline{x}_{d\rightarrow w} + \overline{x}_{e\rightarrow w} & \geq 4 \\
x_{a\rightarrow v} & \geq 0 \\
x_{a\rightarrow w} & \geq 0 \\
x_{e\rightarrow v} & \geq 0 \\
x_{e\rightarrow w} & \geq 0 
\end{align*}
\]

Sum up all constraints & divide by 3 to obtain
Example: Degree Preprocessing with PB Reasoning

\[
\begin{align*}
\overline{x}_{a\rightarrow u} + x_{b\rightarrow v} + x_{b\rightarrow w} &\geq 1 \\
\overline{x}_{a\rightarrow u} + x_{c\rightarrow v} + x_{c\rightarrow w} &\geq 1 \\
\overline{x}_{a\rightarrow u} + x_{d\rightarrow v} + x_{d\rightarrow w} &\geq 1 \\
\overline{x}_{a\rightarrow v} + \overline{x}_{b\rightarrow v} + \overline{x}_{c\rightarrow v} + \overline{x}_{d\rightarrow v} + \overline{x}_{e\rightarrow v} &\geq 4 \\
\overline{x}_{a\rightarrow w} + \overline{x}_{b\rightarrow w} + \overline{x}_{c\rightarrow w} + \overline{x}_{d\rightarrow w} + \overline{x}_{e\rightarrow w} &\geq 4 \\
x_{a\rightarrow v} &\geq 0 \\
x_{a\rightarrow w} &\geq 0 \\
x_{e\rightarrow v} &\geq 0 \\
x_{e\rightarrow w} &\geq 0
\end{align*}
\]

Sum up all constraints & divide by 3 to obtain

\[3\overline{x}_{a\rightarrow u} + 10 \geq 11\]
Example: Degree Preprocessing with PB Reasoning

\[
\begin{align*}
\overline{x}_{a\rightarrow u} + x_{b\rightarrow v} + x_{b\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{c\rightarrow v} + x_{c\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow u} + x_{d\rightarrow v} + x_{d\rightarrow w} & \geq 1 \\
\overline{x}_{a\rightarrow v} + \overline{x}_{b\rightarrow v} + \overline{x}_{c\rightarrow v} + \overline{x}_{d\rightarrow v} + \overline{x}_{e\rightarrow v} & \geq 4 \\
\overline{x}_{a\rightarrow w} + \overline{x}_{b\rightarrow w} + \overline{x}_{c\rightarrow w} + \overline{x}_{d\rightarrow w} + \overline{x}_{e\rightarrow w} & \geq 4 \\
x_{a\rightarrow v} & \geq 0 \\
x_{a\rightarrow w} & \geq 0 \\
x_{e\rightarrow v} & \geq 0 \\
x_{e\rightarrow w} & \geq 0
\end{align*}
\]

Sum up all constraints & divide by 3 to obtain

\[
3\overline{x}_{a\rightarrow u} \geq 1
\]
Example: Degree Preprocessing with PB Reasoning

\begin{align*}
\overline{x}_{a \rightarrow u} + x_{b \rightarrow v} + x_{b \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow u} + x_{c \rightarrow v} + x_{c \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow u} + x_{d \rightarrow v} + x_{d \rightarrow w} & \geq 1 \\
\overline{x}_{a \rightarrow v} + \overline{x}_{b \rightarrow v} + \overline{x}_{c \rightarrow v} + \overline{x}_{d \rightarrow v} + \overline{x}_{e \rightarrow v} & \geq 4 \\
\overline{x}_{a \rightarrow w} + \overline{x}_{b \rightarrow w} + \overline{x}_{c \rightarrow w} + \overline{x}_{d \rightarrow w} + \overline{x}_{e \rightarrow w} & \geq 4 \\
 x_{a \rightarrow v} & \geq 0 \\
x_{a \rightarrow w} & \geq 0 \\
x_{e \rightarrow v} & \geq 0 \\
x_{e \rightarrow w} & \geq 0 \\
\end{align*}

Sum up all constraints & divide by 3 to obtain

\begin{align*}
3\overline{x}_{a \rightarrow u} & \geq 1 \\
\overline{x}_{a \rightarrow u} & \geq 1 \\
\end{align*}
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!


