Pseudo-Boolean Solving and Optimization

Jakob Nordström

University of Copenhagen
and Lund University

“Satisfiability: Theory, Practice, and Beyond” Boot Camp
Simons Institute for the Theory of Computing
February 4, 2021
Organization of This Tutorial

Part I: Pseudo-Boolean Preliminaries

Part II: Pseudo-Boolean Solving

Part III: Pseudo-Boolean Optimization

Part IV: Mixed Integer Linear Programming
Organization of This Tutorial

**Part I:** Pseudo-Boolean Preliminaries

**Part II:** Pseudo-Boolean Solving

**Part III:** Pseudo-Boolean Optimization

**Part IV:** Mixed Integer Linear Programming
Outline of Part I: Pseudo-Boolean Preliminaries

1. Pseudo-Boolean Functions and Constraints
2. Pseudo-Boolean Solving and Optimization
3. Some Further References
Pseudo-Boolean functions:

A pseudo-Boolean function $f : \{0, 1\}^n \to \mathbb{R}$

Studied since 1960s in operations research and 0-1 integer linear programming [BH02]

Restricted versions:
- $f$ represented as polynomial
- $f$ represented as linear form [focus of this tutorial]

Many problems expressible as optimizing value of linear pseudo-Boolean function under linear pseudo-Boolean constraints
Pseudo-Boolean vs. SAT

- Pseudo-Boolean format richer than conjunctive normal form (CNF)

Compare

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

and

$$(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \land (x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_5 \lor x_6) \land (x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \land (x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)$$
Pseudo-Boolean vs. SAT

- Pseudo-Boolean format richer than conjunctive normal form (CNF)

Compare

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]

and

\[
(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \\
\land (x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \\
\land (x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \\
\land (x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \\
\land (x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)
\]

- And pseudo-Boolean reasoning exponentially stronger than conflict-driven clause learning (CDCL)
Pseudo-Boolean vs. SAT

- Pseudo-Boolean format richer than conjunctive normal form (CNF)

Compare

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]

and

\[ (x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \land (x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_5 \lor x_6) \land (x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \land (x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6) \]

- And pseudo-Boolean reasoning exponentially stronger than conflict-driven clause learning (CDCL)
- Yet close enough to SAT to benefit from SAT solving advances
Pseudo-Boolean vs. SAT

- Pseudo-Boolean format richer than conjunctive normal form (CNF)

```
Compare

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]
and

\[
(x_1 \lor x_2 \lor x_3 \lor x_4) \land (x_1 \lor x_2 \lor x_3 \lor x_5) \land (x_1 \lor x_2 \lor x_3 \lor x_6) \\
\land (x_1 \lor x_2 \lor x_4 \lor x_5) \land (x_1 \lor x_2 \lor x_4 \lor x_6) \land (x_1 \lor x_2 \lor x_5 \lor x_6) \\
\land (x_1 \lor x_3 \lor x_4 \lor x_5) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \land (x_1 \lor x_3 \lor x_4 \lor x_6) \\
\land (x_1 \lor x_4 \lor x_5 \lor x_6) \land (x_2 \lor x_3 \lor x_4 \lor x_5) \land (x_2 \lor x_3 \lor x_4 \lor x_6) \\
\land (x_2 \lor x_3 \lor x_5 \lor x_6) \land (x_2 \lor x_4 \lor x_5 \lor x_6) \land (x_3 \lor x_4 \lor x_5 \lor x_6)
\]
```

- And pseudo-Boolean reasoning exponentially stronger than conflict-driven clause learning (CDCL)
- Yet close enough to SAT to benefit from SAT solving advances
- Also possible synergies with 0-1 integer linear programming (ILP)
Pseudo-Boolean Constraints and Normalized Form

In this talk, pseudo-Boolean constraints are $0$-$1$ integer linear constraints

$$\sum \limits_i a_i \ell_i \cong A$$

- $\cong \in \{\geq, \leq, =, >, <\}$
- $a_i, A \in \mathbb{Z}$
- literals $\ell_i$: $x_i$ or $\overline{x_i}$ (where $x_i + \overline{x_i} = 1$)
- variables $x_i$ take values $0 = false$ or $1 = true$
Pseudo-Boolean Constraints and Normalized Form

In this talk, pseudo-Boolean constraints are 0-1 integer linear constraints

$$\sum_i a_i l_i \bowtie A$$

- $\bowtie \in \{\geq, \leq, =, >, <\}$
- $a_i, A \in \mathbb{Z}$
- literals $l_i$: $x_i$ or $\overline{x_i}$ (where $x_i + \overline{x_i} = 1$)
- variables $x_i$ take values $0 = \text{false}$ or $1 = \text{true}$

Convenient to use normalized form [Bar95]

$$\sum_i a_i l_i \geq A$$

- constraint always greater-than-or-equal
- $a_i, A \in \mathbb{N}$
- $A = \deg(\sum_i a_i l_i \geq A)$ referred to as degree (of falsity)
Some Types of Pseudo-Boolean Constraints

Clauses are pseudo-Boolean constraints

\[ x \lor \overline{y} \lor z \iff x + \overline{y} + z \geq 1 \]
Some Types of Pseudo-Boolean Constraints

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

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

2. **Cardinality constraints**

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]
Some Types of Pseudo-Boolean Constraints

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

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

2. **Cardinality constraints**

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \geq 3 \]

3. **General constraints**

\[ x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7 \]
Conversion to Normalized Form: Example

Normalized form used for convenience and without loss of generality

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0\]
Conversion to Normalized Form: Example

Normalized form used for convenience and without loss of generality

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0\]

1. Make inequality non-strict

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1\]
Conversion to Normalized Form: Example

Normalized form used for convenience and without loss of generality

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0\]

1. Make inequality non-strict

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1\]

2. Multiply by \(-1\) to get greater-than-or-equal

\[x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1\]
Conversion to Normalized Form: Example

Normalized form used for convenience and without loss of generality

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0\]

1. Make inequality non-strict

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1\]

2. Multiply by \(-1\) to get greater-than-or-equal

\[x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1\]

3. Replace \(-\ell\) by \(-(1 - \ell)\) [where we define \(\overline{x} \equiv x\)]

\[x_1 - 2(1 - \overline{x}_2) + 3x_3 - 4(1 - \overline{x}_4) + 5x_5 \geq 1\]

\[x_1 + 2\overline{x}_2 + 3x_3 + 4\overline{x}_4 + 5x_5 \geq 7\]
Conversion to Normalized Form: Example

Normalized form used for convenience and without loss of generality

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 < 0\]

1. Make inequality non-strict

\[-x_1 + 2x_2 - 3x_3 + 4x_4 - 5x_5 \leq -1\]

2. Multiply by \(-1\) to get greater-than-or-equal

\[x_1 - 2x_2 + 3x_3 - 4x_4 + 5x_5 \geq 1\]

3. Replace \(-\ell\) by \(-(1 - \ell)\) [where we define \(\overline{x} \equiv x\)]

\[x_1 - 2(1 - \overline{x}_2) + 3x_3 - 4(1 - \overline{x}_4) + 5x_5 \geq 1\]

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

4. Replace "=" by two inequalities "\(\geq\)" and "\(\leq\)"
### Conversion to Normalized Form: Formal Details

Given linear form $\sum_i a_i l_i$ with $\sum_i a_i = M$

<table>
<thead>
<tr>
<th>Syntactic sugar</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_i a_i l_i &gt; A$</td>
<td>$\sum_i a_i l_i \geq A + 1$</td>
</tr>
<tr>
<td>$\sum_i a_i l_i \leq A$</td>
<td>$\sum_i a_i \bar{l}_i \geq M - A$</td>
</tr>
<tr>
<td>$\sum_i a_i l_i &lt; A$</td>
<td>$\sum_i a_i \bar{l}_i \geq M - A + 1$</td>
</tr>
<tr>
<td>$\sum_i a_i l_i = A$</td>
<td>$\sum_i a_i l_i \geq A$ and $\sum_i a_i \bar{l}_i \geq M - A$</td>
</tr>
</tbody>
</table>

In what follows:
- Use syntactic sugar when convenient
- Assume (implicit) normalization whenever it matters
**Linearization**

Possible to **linearize** nonlinear constraints

\[ \sum_{i=1}^{k} a_i m_i \geq A \]

with

\[ m_i = \prod_{j=1}^{d_i} \ell_{i,j} \]
Linearization

Possible to **linearize** nonlinear constraints

\[ \sum_{i=1}^{k} a_i m_i \geq A \]

with

\[ m_i = \prod_{j=1}^{d_i} \ell_{i,j} \]

For instance, using fresh variables \( y_i \) we can write:

\[ \sum_{i=1}^{k} a_i y_i \geq A \]
\[ d_i \cdot \overline{y}_i + \sum_{j=1}^{d_i} \ell_{i,j} \geq d_i \quad i \in [k] \]
\[ y_i + \sum_{j=1}^{d_i} \overline{\ell}_{i,j} \geq 1 \quad i \in [k] \]
Linearization

Possible to **linearize** nonlinear constraints

\[ \sum_{i=1}^{k} a_i m_i \geq A \]

with

\[ m_i = \prod_{j=1}^{d_i} \ell_{i,j} \]

For instance, using fresh variables \( y_i \) we can write:

\[ \sum_{i=1}^{k} a_i y_i \geq A \]

\[ d_i \cdot \overline{y}_i + \sum_{j=1}^{d_i} \ell_{i,j} \geq d_i \quad i \in [k] \]

\[ y_i + \sum_{j=1}^{d_i} \overline{\ell}_{i,j} \geq 1 \quad i \in [k] \]

We won’t go further into this during this talk, though...
Given

- constraints \( C_1 \doteq \sum_i a_i l_i \geq A \) and \( C_2 \doteq \sum_i b_i l_i \geq B \)
- linear form \( L \doteq \sum_i c l_i \)
- positive integer \( k \in \mathbb{N}^+ \)

we will use notation:

\[
C_1 + C_2 \doteq \sum_i (a_i + b_i) \cdot l_i \geq A + B
\]

\[
C_1 + L \doteq \sum_i (a_i + c_i) \cdot l_i \geq A
\]

\[
k \cdot C_1 \doteq \sum_i k a_i \cdot l_i \geq kA
\]

(assuming appropriate normalization whenever needed)
Some Notation for Operations on Constraints (2/2)

Given constraint \( C \implies \sum_i a_i \ell_i \geq A \) with \( \sum_i a_i = M \)

**Negation**

\[ \neg C \implies \sum_i a_i \bar{\ell}_i \geq M - A + 1 \]

**Reification**

\[ z \implies C \implies A \cdot \bar{z} + \sum_i a_i \ell_i \geq A \]

\[ z \iff C \implies (M - A + 1) \cdot z + \sum_i a_i \bar{\ell}_i \geq M - A + 1 \]

\[ z \iff C \implies z \implies C \text{ and } z \iff C \]

**Some calculations**

\[ C + \neg C \implies 0 \geq 1 \]

\[ z \iff C \implies \bar{z} \implies \neg C \]

\[ \deg(C) \cdot (z \geq 1) + (z \implies C) \implies C \]

\[ C + (z \iff C) \implies \deg(\neg C) \cdot z \geq 1 \]
Pseudo-Boolean (PB) formula
Conjunction of pseudo-Boolean constraints
\[ F \equiv C_1 \land C_2 \land \cdots \land C_m \]

Pseudo-Boolean Solving (PBS)
Decide whether \( F \) is satisfiable/feasible

Pseudo-Boolean Optimization (PBO)
Find satisfying assignment to \( F \) that minimizes objective function \( \sum_i w_i \ell_i \)
(Maximization: minimize \( -\sum_i w_i \ell_i \))
Some Problems Expressed as PBO (1/2)

Input:
- undirected graph $G = (V, E)$
- weight function $w : V \rightarrow \mathbb{N}^+$

**Weighted minimum vertex cover**

$$\min \sum_{v \in V} w(v) \cdot x_v$$

$$x_u + x_v \geq 1 \quad (u, v) \in E$$

**Weighted maximum clique**

$$\min - \sum_{v \in V} w(v) \cdot x_v$$

$$\bar{x}_u + \bar{x}_v \geq 1 \quad (u, v) \notin E$$
Some Problems Expressed as PBO (2/2)

Input:
- sets $S_1, \ldots, S_m \subseteq U$
- weight function $w : U \rightarrow \mathbb{N}^+$

Weighted minimum hitting set
Find $H \subseteq U$ such that
- $H \cap S_i \neq \emptyset$ for all $i \in [m]$ ($H$ is a hitting set)
- $\sum_{h \in H} w(h)$ is minimal
Some Problems Expressed as PBO (2/2)

Input:
- sets $S_1, \ldots, S_m \subseteq \mathcal{U}$
- weight function $w : \mathcal{U} \rightarrow \mathbb{N}^+$

Weighted minimum hitting set
Find $H \subseteq \mathcal{U}$ such that
- $H \cap S_i \neq \emptyset$ for all $i \in [m]$ ($H$ is a hitting set)
- $\sum_{h \in H} w(h)$ is minimal

\[
\min \sum_{e \in \mathcal{U}} w(e) \cdot x_e \\
\sum_{e \in S_i} x_e \geq 1 \quad i \in [m]
\]
Some Problems Expressed as PBO (2/2)

Input:
- sets $S_1, \ldots, S_m \subseteq \mathcal{U}$
- weight function $w : \mathcal{U} \rightarrow \mathbb{N}^+$

**Weighted minimum hitting set**

Find $H \subseteq \mathcal{U}$ such that
- $H \cap S_i \neq \emptyset$ for all $i \in [m]$ ($H$ is a hitting set)
- $\sum_{h \in H} w(h)$ is minimal

$$
\min \sum_{e \in \mathcal{U}} w(e) \cdot x_e \\
\sum_{e \in S_i} x_e \geq 1 \quad i \in [m]
$$

Note: In all of these examples, the problem is to
- optimize a linear function
- subject to a CNF formula (all constraints are clausal)

Already expressive framework!
What we will discuss in this tutorial:

1. Pseudo-Boolean (PB) solving and optimization [main focus]
2. MaxSAT solving
3. Integer linear programming (ILP) — or, more generally, mixed integer linear programming (MIP)
Approaches for Pseudo-Boolean Problems

What we will discuss in this tutorial:

1. Pseudo-Boolean (PB) solving and optimization [main focus]
2. MaxSAT solving
3. Integer linear programming (ILP) — or, more generally, mixed integer linear programming (MIP)

Rough conceptual difference:

- **PB/SAT**: Focus on integral solutions, try to find optimal one
- **ILP/MIP**: Find optimal non-integer solution; search for integral solutions nearby

Basic trade-off: Inference power vs. inference speed
Some References for Further Reading (and Watching)

**Handbook of Satisfiability (PB and MaxSAT)**
- Chapter 7: Proof Complexity and SAT Solving
- Chapter 23: MaxSAT, Hard and Soft Constraints
- Chapter 24: Maximum Satisfiability
- Chapter 28: Pseudo-Boolean and Cardinality Constraints

**Mixed integer linear programming**
- [Wol08](https://tinyurl.com/MIPsurveypaper)
- [KMP13](https://tinyurl.com/MIPperformance)

**Videos**
- MaxSAT tutorial by Berg et al. [https://tinyurl.com/MaxSATtutorial](https://tinyurl.com/MaxSATtutorial)
- MIP tutorial by Gleixner [https://tinyurl.com/MIPtutorial](https://tinyurl.com/MIPtutorial)
Organization of This Tutorial

Part I: Pseudo-Boolean Preliminaries

Part II: Pseudo-Boolean Solving

Part III: Pseudo-Boolean Optimization

Part IV: Mixed Integer Linear Programming
Outline of Part II: Pseudo-Boolean Solving

4 Conflict-Driven Clause Learning
   - CDCL by Example
   - Pseudocode and Analysis

5 CDCL-Based Pseudo-Boolean Solving
   - Some Example CNF Encodings
   - Properties of CNF Encodings

6 “Native” Cutting-Planes-Based Pseudo-Boolean Solving
   - Preliminaries on Pseudo-Boolean Reasoning
   - Pseudo-Boolean Conflict Analysis Using Saturation
   - Pseudo-Boolean Conflict Analysis Using Division
   - More About Pseudo-Boolean Reasoning
A Quick Recap of Modern SAT Solving

**DPLL method** [DP60, DLL62]

- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause
A Quick Recap of Modern SAT Solving

**DPLL method** [DP60, DLL62]
- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

**Conflict-driven clause learning (CDCL)** [MS96, BS97, MMZ+01]
- Analyse conflicts in more detail — add new clauses to formula
- More efficient backtracking
- Also let conflicts guide other heuristics
A Quick Recap of Modern SAT Solving

**DPLL method** [DP60, DLL62]
- Assign values to variables (in some smart way)
- Backtrack when conflict with falsified clause

**Conflict-driven clause learning (CDCL)** [MS96, BS97, MMZ+01]
- Analyse conflicts in more detail — add new clauses to formula
- More efficient backtracking
- Also let conflicts guide other heuristics
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\((u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (u \lor w) \land (\overline{u} \lor \overline{w})\)

**Decision**

Free choice to assign value to variable

**Notation**  \(w \overset{d}{=} 0\)
Variable Assignments

Two kinds of assignments — illustrate on example formula:

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

Decision

Free choice to assign value to variable

Notation $w^d = 0$
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[w^d = 0\]

**Decision**

Free choice to assign value to variable

Notation \( w^d = 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

Given \( w = 0 \), clause \( \overline{u} \lor w \) forces \( u = 0 \)

Notation \( u^{\overline{u} \lor w} = 0 \) (\( \overline{u} \lor w \) is reason)
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

**Decision**

Free choice to assign value to variable

Notation \( w \overset{d}{=} 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

Given \( w = 0 \), clause \( \overline{u} \lor w \) forces \( u = 0 \)

Notation \( u \overset{\text{forced}}{=} 0 \) (\( \overline{u} \lor w \) is reason)
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[\begin{align*}
w \doteq 0 \\
u \overline{u} \lor w \doteq 0 \\
x \doteq 0
\end{align*}\]

**Decision**
Free choice to assign value to variable

**Notation** \(w \doteq 0\)

**Unit propagation**
Forced choice to avoid falsifying clause
Given \(w = 0\), clause \(\overline{u} \lor w\) forces \(u = 0\)

**Notation** \(u \overline{u} \lor w \doteq 0\) (\(\overline{u} \lor w\) is reason)

Always propagate if possible, otherwise decide
Until satisfying assignment or conflict clause
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
w &\overset{d}{=} 0 \\
\overline{u} \lor w &\overset{d}{=} 0 \\
u &\overset{d}{=} 0 \\
u \lor x \lor y &\overset{d}{=} 1
\end{align*}
\]

**Decision**

Free choice to assign value to variable

Notation \( w \overset{d}{=} 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

Given \( w = 0 \), clause \( \overline{u} \lor w \) forces \( u = 0 \)

Notation \( u \overset{d}{=} 0 \) (\( \overline{u} \lor w \) is reason)

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[w^d = 0\]

\[u \overline{u} \lor w = 0\]

\[x^d = 0\]

\[y \overline{u} \lor x \lor y = 1\]

\[z \overline{x} \lor \overline{y} \lor z = 1\]

**Decision**
Free choice to assign value to variable

**Notation** \(w^d = 0\)

**Unit propagation**
Forced choice to avoid falsifying clause
Given \(w = 0\), clause \(\overline{u} \lor w\) forces \(u = 0\)

**Notation** \(u \overline{u} \lor w = 0\) (\(\overline{u} \lor w\) is reason)

Always propagate if possible, otherwise decide
Until satisfying assignment or conflict clause
Variable Assignments

Two kinds of assignments — illustrate on example formula:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

**Decision**

Free choice to assign value to variable

Notation \( w \vdash 0 \)

**Unit propagation**

Forced choice to avoid falsifying clause

Given \( w = 0 \), clause \( \overline{u} \lor w \) forces \( u = 0 \)

Notation \( u \vdash 0 \) (\( \overline{u} \lor w \) is reason)

Always propagate if possible, otherwise decide

Until satisfying assignment or conflict clause
Conflict-Driven Clause Learning

Time to analyse this conflict!

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[w^d = 0\]

\[u \lor w = 0\]

\[x^d = 0\]

\[y = 1\]

\[z = 1\]

\[\overline{y} \lor \overline{z}\]
Conflict-Driven Clause Learning

Time to analyse this conflict!

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
    w & = 0 \\
    u & = 0 \\
    x & = 0 \\
    u & = 1 \\
    y & = 1 \\
    z & = 1 \\
    \overline{y} & = 0
\end{align*}
\]

Could backtrack by flipping last decision

Jakob Nordström (UCPH & LU)  Pseudo-Boolean Solving and Optimization  Simons Institute Feb '21
Conflict-Driven Clause Learning

Time to analyse this conflict!

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{array}{c}
\text{\d{w} \equiv 0} \\
\text{\d{u} \equiv 0} \\
\text{\d{x} \equiv 0} \\
\text{\d{y} \equiv 1} \\
\text{\d{z} \equiv 1} \\
\text{\d{y} \lor \overline{z} \bot}
\end{array}
\]

Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible
Conflict-Driven Clause Learning

Time to analyse this conflict!

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over \(z\) for last two clauses:

- \(x \lor \overline{y} \lor z\) wants \(z = 1\)
- \(\overline{y} \lor \overline{z}\) wants \(z = 0\)

Merge & remove \(z\) — must satisfy \(x \lor \overline{y}\)
Time to analyse this conflict!

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Could backtrack by flipping last decision

But want to learn from conflict and cut away as much of search space as possible

Case analysis over \(z\) for last two clauses:

- \(x \lor \overline{y} \lor z\) wants \(z = 1\)
- \(\overline{y} \lor \overline{z}\) wants \(z = 0\)

Merge & remove \(z\) — must satisfy \(x \lor \overline{y}\)

Repeat until only 1 variable after last decision — learn that clause (1UIP) and backjump
Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \neg y \lor z) \land (\neg x \lor z) \land (\neg y \lor \neg z) \land (\neg x \lor \neg z) \land (\neg u \lor w) \land (\neg u \lor \neg w)\]

\[
\begin{align*}
  w^d &= 0 \\
  u^u \lor w &= 0 \\
  x^u &= 0 \\
  y^u \lor x \lor y &= 1 \\
  z^u \lor \neg y \lor z &= 1 \\
  \neg y \lor \neg z &= \perp
\end{align*}
\]
Complete Example of CDCL Execution

**Backjump:** roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
  w^d & = 0 \\
  u^d & = 0 \\
  x^d & = 0 \\
  y^d & = 1 \\
  z^d & = 1 \\
  y^d & = 1
\end{align*}
\]
Complete Example of CDCL Execution

**Backjump:** roll back max #decisions so that last variable still flips

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

```
 w^d = 0

 u \lor w = 0

 u \lor w = 0

 x = 0

 x = 1

 x = 1

 y = 1

 z = 1

 z = 1
```

```
 u \lor x

 x \lor \overline{y}

 \overline{y} \lor \overline{z}
```
Complete Example of CDCL Execution

**Backjump:** roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Complete Example of CDCL Execution

**Backjump:** roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (u \lor w) \land (\overline{u} \lor \overline{w})\]

\[d_w = 0\]  
\[u \stackrel{d}{=} 0\]  
\[x \stackrel{d}{=} 0\]  
\[y \stackrel{d}{=} 1\]  
\[\overline{y} \stackrel{d}{=} 1\]  
\[z \stackrel{d}{=} 1\]  
\[\overline{x} \perp\]  
\[\overline{x} \perp\]
Complete Example of CDCL Execution

Backjump: roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Complete Example of CDCL Execution

**Backjump:** roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

- \[w^d = 0\]
- \[u = 0\]
- \[x^d = 0\]
- \[y = 1\]
- \[z = 1\]
- \[\overline{y} \lor \overline{z} = 1\]
- \[\overline{x} = 0\]
**Complete Example of CDCL Execution**

**Backjump:** roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
    w^d & = 0 \\
    u & = 0 \\
    x & = 0 \\
    y & = 1 \\
    z & = 1 \\
    \overline{y} \lor \overline{z} & = \bot
\end{align*}
\]

\[
\begin{align*}
    w^d & = 0 \\
    \overline{u} \lor w & = 0 \\
    u & = 0 \\
    x & = 1 \\
    \overline{u} \lor w & = 1 \\
    \overline{y} \lor \overline{z} & = \bot
\end{align*}
\]
Complete Example of CDCL Execution

Backjump: roll back max \#decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (y \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Backjump: roll back max #decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
Complete Example of CDCL Execution

**Backjump:** roll back max \#decisions so that last variable still flips

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]
CDCL Main Loop Pseudocode (High Level)

```plaintext
forever do
    if current assignment falsifies clause then
        apply learning scheme to derive new clause;
        if learned clause empty then output UNSATISFIABLE and exit;
        else
            add learned clause and backjump
        end
    else if all variables assigned then output SATISFIABLE and exit;
    else if exists unit clause \( C \) propagating \( x \) to value \( b \in \{0, 1\} \) then
        add propagated assignment \( x = b \)
    else if time to restart then
        remove all variable assignments
    else
        if time for clause database reduction then
            erase (roughly) half of learned clauses in memory
        end
        use decision scheme to choose assignment \( x^d = b \);
    end
end
```
CDCL Main Loop Pseudocode (High Level)

\[
\text{forever do}
\quad \text{if current assignment falsifies clause then}
\quad \quad \text{apply learning scheme to derive new clause;}
\quad \quad \text{if learned clause empty then output UNSATISFIABLE and exit;}
\quad \quad \text{else}
\quad \quad \quad \text{add learned clause and backjump}
\quad \text{end}
\quad \text{else if all variables assigned then output SATISFIABLE and exit;}
\quad \text{else if exists unit clause } C \text{ propagating } x \text{ to value } b \in \{0, 1\} \text{ then}
\quad \quad \text{add propagated assignment } x \overset{d}{=} b
\quad \text{else if time to restart then}
\quad \quad \text{remove all variable assignments}
\quad \text{else}
\quad \quad \text{if time for clause database reduction then}
\quad \quad \quad \text{erase (roughly) half of learned clauses in memory}
\quad \quad \text{end}
\quad \quad \text{use decision scheme to choose assignment } x \overset{d}{=} b;
\quad \text{end}
\text{end}
\]
CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Best(?) rigorous method: Focus on underlying method of reasoning
CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Best(?) rigorous method: Focus on underlying method of reasoning

Resolution proof system
- Start with clauses of formula
- Derive new clauses by resolution rule

\[
\frac{C \lor x \quad D \lor \overline{x}}{C \lor D}
\]

- Done when contradiction $\bot$ in form of empty clause derived
CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Best(?) rigorous method: Focus on underlying method of reasoning

Resolution proof system
- Start with clauses of formula
- Derive new clauses by resolution rule

$$
\frac{C \lor x}{C \lor \bar{x}}
\quad
\frac{D \lor \bar{x}}{D
\}

\quad
\frac{C \lor \bar{x}}{C \lor D}
$$

- Done when contradiction $\bot$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*
So lower bounds on proof size $\Rightarrow$ lower bounds on running time
CDCL Analysis and the Resolution Proof System

How to analyse CDCL performance?
Many intricate, hard-to-understand heuristics
Best(?) rigorous method: Focus on underlying method of reasoning

Resolution proof system
- Start with clauses of formula
- Derive new clauses by resolution rule

\[
\frac{C \lor x}{C \lor \overline{x}} \quad \frac{D \lor \overline{x}}{D}
\]

- Done when contradiction $\bot$ in form of empty clause derived

When run on unsatisfiable formula, CDCL generates resolution proof*
So lower bounds on proof size $\Rightarrow$ lower bounds on running time

(*) Ignores preprocessing, but we don’t have time to go into this
Resolution Proofs from CDCL Executions

Obtain resolution proof...
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution...
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:

\[
\begin{align*}
  w^d &= 0 \\
  u^\bot &\lor w \\
  x^d &= 0 \\
  u &\lor w = 0 \\
  x &\lor \overline{y} \lor z \\
  \overline{y} &\lor \overline{z} \\
  x &\lor \overline{y} \\
  \overline{x} &\lor \overline{z} \\
  \overline{u} &\lor w \\
  x &\lor \overline{u} \\
  \bot
\end{align*}
\]
Resolution Proofs from CDCL Executions

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:
Current State of Affairs

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")

- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?

- Paradox: resolution quite weak proof system; many strong lower bounds for "obvious" formulas, e.g., [Hak85, Urq87, BW01, MN14]

- Explore stronger reasoning methods (potential exponential speed-up)

- In particular, pseudo-Boolean solving (a.k.a. 0-1 integer programming) corresponding to cutting planes proof system

- Importantly, extends to pseudo-Boolean optimization [we will return to this topic in Part III]
Approaches to Pseudo-Boolean Solving

Conversion to disjunctive clauses

- Lazy approach: learn clauses from PB constraints
  - \texttt{SAT4J} [LP10] (one of versions in library)
Approaches to Pseudo-Boolean Solving

Conversion to disjunctive clauses

- Lazy approach: learn clauses from PB constraints
  - Sat4J [LP10] (one of versions in library)
- Eager approach: re-encode to clauses and run CDCL
  - MiniSat+ [ES06]
  - Open-WBO [MML14]
  - NaPS [SN15]
Conversion to disjunctive clauses

- Lazy approach: learn clauses from PB constraints
  - Sat4J [LP10] (one of versions in library)
- Eager approach: re-encode to clauses and run CDCL
  - MiniSat+ [ES06]
  - Open-WBO [MML14]
  - NAPS [SN15]

Native reasoning with pseudo-Boolean constraints

- PRS [DG02]
- Galena [CK05]
- Pueblo [SS06]
- Sat4J [LP10]
- RoundingSat [EN18]
Re-encoding to CNF

- CNF encoding can be exponentially larger than PB encoding
- Use extension variables for more compact encoding
- High-level idea: new variables = gates in circuit evaluating PB constraint
- Consider first two concrete examples for cardinality constraints

\[ \sum_{i=1}^{n} x_i \otimes k \]  

(\text{where } \otimes \in \{\geq, \leq, =\})
Sequential Counter Encoding

\[ \sum_{i=1}^{n} x_i \otimes k \text{ for } \otimes \in \{ \geq, \leq, = \} \]

\[ s_{i,j} = \text{“sum of } i \text{ first variables } \geq j” \] (from [Sin05] with slight twists)
Sequential Counter Encoding

\[ \sum_{i=1}^{n} x_i \otimes k \text{ for } \otimes \in \{\geq, \leq, =\} \]

\( s_{i,j} = \text{“sum of } i \text{ first variables } \geq j\)" (from [Sin05] with slight twists)

**Base case (} j > 1):**

\[
\begin{align*}
\bar{x}_1 & \lor s_{1,1} \\
\bar{s}_{1,j} & \\
x_1 & \lor \bar{s}_{1,1}
\end{align*}
\]

**Inductive step (} i \geq 2, } j \geq 1):**

\[
\begin{align*}
\bar{x}_i & \lor s_{i,1} \\
\bar{s}_{i-1,j} & \lor s_{i,j} \\
x_i & \lor \bar{s}_{i-1,j} \lor s_{i,j+1} \\
x_i & \lor s_{i-1,j+1} \lor \bar{s}_{i,j+1} \\
\bar{s}_{i-1,j} & \lor \bar{s}_{i-1,j+1} \lor \bar{s}_{i,j+1}
\end{align*}
\]
Sequential Counter Encoding

\[ \sum_{i=1}^{n} x_i \bowtie k \text{ for } \bowtie \in \{\geq, \leq, =\} \]

\( s_{i,j} = "\text{sum of } i \text{ first variables } \geq j" \) (from [Sin05] with slight twists)

**Base case** (\( j > 1 \)):

\[
\overline{x}_1 \lor s_{1,1} \\
\overline{s}_{1,j} \\
x_1 \lor \overline{s}_{1,1}
\]

**Inductive step** (\( i \geq 2, j \geq 1 \)):

\[
\overline{x}_i \lor s_{i,1} \\
\overline{s}_{i-1,j} \lor s_{i,j} \\
x_i \lor \overline{s}_{i-1,j} \lor s_{i,j+1} \\
x_i \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1} \\
s_{i-1,j} \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1}
\]

To enforce cardinality constraint

- \( \bowtie \vdash \geq \): Add unit clause \( s_{n,k} \)
- \( \bowtie \vdash \leq \): Add unit clause \( \overline{s}_{n,k+1} \)
- \( \bowtie \vdash = \): Add both unit clauses above
Totalizer Encoding

\[ \sum_{i=1}^{n} x_i \otimes k \text{ for } \otimes \in \{\geq, \leq, =\} \]

Build binary tree: children have \( t \) bits \( a_i, b_i \) each; parent outputs \( 2t \) bits \( c_j \)

\( c_j = \text{"sum of input variables } \geq j\" \) [BB03]
Totalizer Encoding

\[ \sum_{i=1}^{n} x_i \bowtie k \text{ for } \bowtie \in \{ \geq, \leq, = \} \]

Build binary tree: children have \( t \) bits \( a_i, b_i \) each; parent outputs \( 2t \) bits \( c_j \)

\( c_j = \text{“sum of input variables } \geq j \text{”} \) [BB03]

**Base case (two bits} \( x_1, x_2 \):**
- \( \overline{x}_i \lor c_1 \)
- \( \overline{x}_1 \lor \overline{x}_2 \lor c_2 \)
- \( x_1 \lor x_2 \lor \overline{c}_1 \)
- \( x_i \lor \overline{c}_2 \)

**Inductive step (} \( i + j \geq 1 \):**
- \( \overline{a}_i \lor \overline{b}_j \lor c_{i+j} \)
- \( a_{i+1} \lor b_{j+1} \lor \overline{c}_{i+j+1} \)
- \( (a_0 = b_0 = 1) \)
Totalizer Encoding

$$\sum_{i=1}^{n} x_i \bowtie k \text{ for } \bowtie \in \{\geq, \leq, =\}$$

Build binary tree: children have $t$ bits $a_i, b_i$ each; parent outputs $2t$ bits $c_j$

$c_j = \text{“sum of input variables } \geq j\text{”}$ [BB03]

**Base case (two bits } x_1, x_2):**

- $\bar{x}_i \lor c_1$
- $\bar{x}_1 \lor \bar{x}_2 \lor c_2$
- $x_1 \lor x_2 \lor \bar{c}_1$
- $x_i \lor \bar{c}_2$

**Inductive step (} i + j \geq 1):**

- $\bar{a}_i \lor \bar{b}_j \lor c_{i+j}$
- $a_{i+1} \lor b_{j+1} \lor \bar{c}_{i+j+1}$
- $(a_0 = b_0 = 1)$

To enforce cardinality constraint, add for root node

- $\bowtie = \geq$: unit clause $c_k$
- $\bowtie = \leq$: unit clause $\bar{c}_{k+1}$
- $\bowtie = =$: both unit clauses above

Can be extended to arbitrary PB constraints [JMM15]; blow-up can be bad
Adder Network Encoding (Sketch)

- For general pseudo-Boolean constraints $\sum_{i=1}^{n} a_i l_i \geq A$, write coefficients $a_i$ in binary $\langle a_i, B a_i, B - 1 \cdots a_i, 1 a_i, 0 \rangle$
Adder Network Encoding (Sketch)

- For general pseudo-Boolean constraints $\sum_{i=1}^{n} a_i l_i \geq A$, write coefficients $a_i$ in binary $\langle a_i, B a_i, B-1 \cdots a_i, 1 a_i, 0 \rangle$
- Assuming $B$ large enough for rest of this slide, it clearly holds that

$$\sum_{i=1}^{n} a_i l_i = \sum_{i=1}^{n} \sum_{j=0}^{B} 2^j \cdot a_{i,j} \cdot l_i$$
Adder Network Encoding (Sketch)

- For general pseudo-Boolean constraints $\sum_{i=1}^{n} a_i \ell_i \geq A$, write coefficients $a_i$ in binary $\langle a_i, B a_i, B-1 \cdots a_i, 1 a_i, 0 \rangle$

- Assuming $B$ large enough for rest of this slide, it clearly holds that

$$\sum_{i=1}^{n} a_i \ell_i = \sum_{i=1}^{n} \sum_{j=0}^{B} 2^j \cdot a_{i,j} \cdot \ell_i$$

- Introduce new variables $c_{out}$, $s_{out}$ and use encodings of full adders

$$2 \cdot c_{out} + s_{out} = x + y + z$$

in CNF to enforce

$$\sum_{i=1}^{n} \sum_{j=0}^{B} 2^j \cdot a_{i,j} \cdot \ell_i = \sum_{j=0}^{B} 2^j \cdot s_j \cdot \ell_i$$

and

$$\sum_{j=0}^{B} 2^j \cdot s_j \cdot \ell_i \geq A$$
Adder Network Encoding (Sketch)

- For general pseudo-Boolean constraints $\sum_{i=1}^{n} a_i \ell_i \geq A$, write coefficients $a_i$ in binary $\langle a_{i,B} a_{i,B-1} \cdots a_{i,1} a_{i,0} \rangle$

- Assuming $B$ large enough for rest of this slide, it clearly holds that

$$\sum_{i=1}^{n} a_i \ell_i = \sum_{i=1}^{n} \sum_{j=0}^{B} 2^j \cdot a_{i,j} \cdot \ell_i$$

- Introduce new variables $c_{\text{out}}, s_{\text{out}}$ and use encodings of full adders

$$2 \cdot c_{\text{out}} + s_{\text{out}} = x + y + z$$

in CNF to enforce

$$\sum_{i=1}^{n} \sum_{j=0}^{B} 2^j \cdot a_{i,j} \cdot \ell_i = \sum_{j=0}^{B} 2^j \cdot s_j \cdot \ell_i \quad \text{and} \quad \sum_{j=0}^{B} 2^j \cdot s_j \cdot \ell_i \geq A$$

- See [ES06] for all the missing details...
CNF Encoding Desiderata

**Generalized arc consistency (GAC)**
For $F_C$ encoding PB constraint $C$ and $\rho$ partial assignment, want:
- If $C$ propagates under $\rho$, then $F_C$ should yield same propagations
- If $\rho$ falsifies $C$, then $F_C$ should unit propagate to contradiction

True for sequential counter and totalizer; false for adder network
CNF Encoding Desiderata

**Generalized arc consistency (GAC)**
For $F_C$ encoding PB constraint $C$ and $\rho$ partial assignment, want:
- If $C$ propagates under $\rho$, then $F_C$ should yield same propagations
- If $\rho$ falsifies $C$, then $F_C$ should unit propagate to contradiction

True for sequential counter and totalizer; false for adder network

**Encoding size**
Want as few variables and clauses as possible

Adder network very compact
Totalizer has fewer variables than sequential counter
But generalized totalizer encoding can get exponentially large
CNF Encoding Desiderata

**Generalized arc consistency (GAC)**

For $F_C$ encoding PB constraint $C$ and $\rho$ partial assignment, want:

- If $C$ propagates under $\rho$, then $F_C$ should yield same propagations
- If $\rho$ falsifies $C$, then $F_C$ should unit propagate to contradiction

True for sequential counter and totalizer; false for adder network

**Encoding size**

Want as few variables and clauses as possible

Adder network very compact

Totalizer has fewer variables than sequential counter

But generalized totalizer encoding can get exponentially large

Possible to achieve both GAC and polynomial-size encoding [BBR09]

But complicated; and in practice not better than totalizer [JMM15]?

Rich literature on encodings — see SAT handbook for more references
Performance of CDCL-Based Pseudo-Boolean Solving

- CDCL-based pseudo-Boolean can be very competitive (sometimes beating native pseudo-Boolean solvers hands down)
- Extension variables potentially gives solver lots of power
  - Allows branching over complex statements
  - Can learn clauses corresponding to polytopes in original problem
- But performance gain from extension variables seems quite sensitive to input order [EGNV18]
- And sometimes extension variables cannot make up for CDCL being exponentially weaker than pseudo-Boolean reasoning [EGNV18]
Question About Forward vs. Backward Propagation

- **Forward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ true, then $s_{n,k} / c_k$ propagates to true
- **Backward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ false, then $s_{n,k} / c_k$ propagates to false

**Sequential counter**

\[
\overline{x}_i \lor s_{i,1}
\]

\[
\overline{s}_{i-1,j} \lor s_{i,j}
\]

\[
\overline{x}_i \lor \overline{s}_{i-1,j} \lor s_{i,j+1}
\]

\[
x_i \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1}
\]

\[
s_{i-1,j} \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1}
\]

**Totalizer**

\[
\overline{a}_i \lor \overline{b}_j \lor c_{i+j}
\]

\[
a_{i+1} \lor b_{j+1} \lor \overline{c}_{i+j+1}
\]
Question About Forward vs. Backward Propagation

- **Forward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ true, then $s_{n,k} / c_k$ propagates to true
- **Backward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ false, then $s_{n,k} / c_k$ propagates to false

**Sequential counter**

\[
\begin{align*}
\overline{x}_i & \lor s_{i,1} \\
\overline{s}_{i-1,j} & \lor s_{i,j} \\
\overline{x}_i & \lor \overline{s}_{i-1,j} \lor s_{i,j+1} \\
x_i & \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1} \\
s_{i-1,j} & \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1}
\end{align*}
\]

**Totalizer**

\[
\begin{align*}
\overline{a}_i & \lor \overline{b}_j \lor c_{i+j} \\
a_{i+1} & \lor b_{j+1} \lor \overline{c}_{i+j+1}
\end{align*}
\]
Question About Forward vs. Backward Propagation

- **Forward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ true, then $s_{n,k} / c_k$ propagates to true
- **Backward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ false, then $s_{n,k} / c_k$ propagates to false

**Sequential counter**

\[
\begin{align*}
\overline{x}_i & \lor s_{i,1} \\
\overline{s}_{i-1,j} & \lor s_{i,j} \\
\overline{x}_i & \lor \overline{s}_{i-1,j} \lor s_{i,j+1} \\
x_i & \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1} \\
s_{i-1,j} & \lor s_{i-1,j+1} \lor \overline{s}_{i,j+1}
\end{align*}
\]

**Totalizer**

\[
\begin{align*}
\overline{a}_i & \lor \overline{b}_j \lor c_{i+j} \\
a_{i+1} & \lor b_{j+1} \lor \overline{c}_{i+j+1}
\end{align*}
\]
Question About Forward vs. Backward Propagation

- **Forward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ true, then $s_{n,k} / c_k$ propagates to true
- **Backward propagation**: If $\sum_{i=1}^{n} x_i \geq k$ false, then $s_{n,k} / c_k$ propagates to false

### Sequential counter

- $\overline{x_i} \lor s_{i,1}$
- $\overline{s_{i-1,j}} \lor s_{i,j}$
- $\overline{x_i} \lor \overline{s_{i-1,j}} \lor s_{i,j+1}$
- $x_i \lor s_{i-1,j+1} \lor \overline{s_{i,j+1}}$
- $s_{i-1,j} \lor s_{i-1,j+1} \lor \overline{s_{i,j+1}}$

### Totalizer

- $\overline{a_i} \lor \overline{b_j} \lor c_{i+j}$
- $a_{i+1} \lor b_{j+1} \lor \overline{c_{i+j+1}}$

Solvers like **OPEN-WBO** [MML14] only encode forward propagation

- Can having propagation in both directions help?
- Or does it on the contrary hurt? Why?
More Questions

1. How to find best possible CNF encodings of PB constraints for given problem?
   - Trade-offs between propagation strength and encoding size?
   - Rigorous mathematical insights?

2. Understand complementary strengths of CDCL-based and “native” cutting-planes-based PB solving?
   - Theoretical results on computational complexity?
   - Harness complementary strengths in applied solvers?

3. How to make sure re-encoding into CNF is guaranteed to be correct?
“Native” Pseudo-Boolean Conflict-Driven Search

Want to do “same thing” as CDCL but with pseudo-Boolean constraints without re-encoding

- Variable assignments
  1. Always propagate forced assignment if possible
  2. Otherwise make assignment using decision heuristic

- At conflict
  1. Do conflict analysis to derive new constraint
  2. Add new constraint to instance
  3. Backjump by rolling back decisions so that asserting literal flips
Propagations, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered}) \text{ set of literals assigned true}\}$
Propagation, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{\text{(ordered) set of literals assigned true}\}$

**Slack** measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$slack(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$
Propagating, Conflict, and Slack

Let \( \rho \) current assignment of solver (a.k.a. trail)
Represent as \( \rho = \{ \text{(ordered) set of literals assigned true} \} \)

Slack measures how far \( \rho \) is from falsifying \( \sum_i a_i \ell_i \geq A \)

\[
\text{slack}(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A
\]

Consider \( C : x_1 + 2\bar{x}_2 + 3x_3 + 4\bar{x}_4 + 5x_5 \geq 7 \)

<table>
<thead>
<tr>
<th>( \rho )</th>
<th>( \text{slack}(C; \rho) )</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>( {x_5} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {x_5, x_4} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( {x_5, x_4, x_3, x_2} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note that constraint can be conflicting though not all variables assigned
Propagation, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered}) \text{ set of literals assigned true}\}$

Slack measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$\text{slack}(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$

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

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\text{slack}(C; \rho)$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
# Propagation, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered}) \text{ set of literals assigned true}\}$

Slack measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$\text{slack}(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$

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

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\text{slack}(C; \rho)$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>${\overline{x}_5}$</td>
<td>3</td>
<td>propagates $\overline{x}_4$ (coefficient $&gt; \text{slack}$)</td>
</tr>
</tbody>
</table>
Propagation, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered}) \text{ set of literals assigned true}\}$

Slack measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$slack(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$

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

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$slack(C; \rho)$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>${}$</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>${\overline{x}_5}$</td>
<td>3</td>
<td>propagates $\overline{x}_4$ (coefficient $&gt; \text{slack}$)</td>
</tr>
<tr>
<td>${\overline{x}_5, \overline{x}_4}$</td>
<td>3</td>
<td>propagation doesn’t change slack</td>
</tr>
</tbody>
</table>
Propagation, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered}) \text{ set of literals assigned true}\}$

**Slack** measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$\text{slack}(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$

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

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$\text{slack}(C; \rho)$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>${}$</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>${\overline{x}_5}$</td>
<td>3</td>
<td>propagates $\overline{x}_4$ (coefficient $&gt; \text{slack}$)</td>
</tr>
<tr>
<td>${\overline{x}_5, \overline{x}_4}$</td>
<td>3</td>
<td>propagation doesn’t change slack</td>
</tr>
<tr>
<td>${\overline{x}_5, \overline{x}_4, \overline{x}_3, x_2}$</td>
<td>$-2$</td>
<td>conflict ($\text{slack} &lt; 0$)</td>
</tr>
</tbody>
</table>
Propagating, Conflict, and Slack

Let $\rho$ current assignment of solver (a.k.a. trail)
Represent as $\rho = \{(\text{ordered})\ \text{set of literals assigned true}\}$

**Slack** measures how far $\rho$ is from falsifying $\sum_i a_i \ell_i \geq A$

$$slack(\sum_i a_i \ell_i \geq A; \rho) = \sum_{\ell_i \text{ not falsified by } \rho} a_i - A$$

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

<table>
<thead>
<tr>
<th>$\rho$</th>
<th>$slack(C; \rho)$</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>${\overline{x}_5}$</td>
<td>3</td>
<td>propagates $\overline{x}_4$ (coefficient $&gt; \text{slack}$)</td>
</tr>
<tr>
<td>${\overline{x}_5, \overline{x}_4}$</td>
<td>3</td>
<td>propagation doesn’t change slack</td>
</tr>
<tr>
<td>${\overline{x}_5, \overline{x}_4, \overline{x}_3, x_2}$</td>
<td>$-2$</td>
<td>conflict (slack $&lt; 0$)</td>
</tr>
</tbody>
</table>

Note that constraint can be conflicting though not all variables assigned
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again

\((u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\)

\[
\begin{align*}
\text{Assignments} & \\
\overline{w} & = 0 \\
\overline{u} \lor w & = 0 \\
x & = 0 \\
u \lor x \lor y & = 1 \\
y & = 1 \\
\overline{x} \lor \overline{y} \lor z & = 1 \\
\overline{y} \lor \overline{z} & \\
\end{align*}
\]
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Assignment “left on trail” always falsifies derived clause
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
  w^d &= 0 \\
  \overline{u} \lor \overline{w} &= 0 \\
  x^d &= 0 \\
  y^\overline{x} \lor y^\overline{y} &= 1 \\
  \overline{x} \lor \overline{\overline{y}} \lor \overline{z} &= 1 \\
  \overline{y} \lor \overline{z} &\text{ falsified by} \\
  \text{trail } \rho &= \{ \overline{w}, \overline{u}, \overline{x}, y, z \}
\end{align*}
\]

Assignment “left on trail” always falsifies derived clause.
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Assignment "left on trail" always falsifies derived clause.
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Assignment “left on trail” always falsifies derived clause
Conflicts Analysis Invariant

Look at our example CDCL conflict analysis again

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

Assignment "left on trail" always falsifies derived clause

\[\Rightarrow\] every derived constraint "explains" conflict
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again:

\[(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})\]

\[
\begin{align*}
  w & \equiv 0 \\
  u & \equiv 0 \\
  x & \equiv 0 \\
  y & \equiv 1 \\
  z & \equiv 1 \\
  \overline{y} \lor \overline{z} & \perp
\end{align*}
\]

Assignment “left on trail” always falsifies derived clause

\[
\Rightarrow \text{every derived constraint “explains” conflict}
\]

Terminate conflict analysis when explanation looks nice

Assignment “left on trail” always falsifies derived clause

⇒ every derived constraint “explains” conflict

Terminate conflict analysis when explanation looks nice
Conflict Analysis Invariant

Look at our example CDCL conflict analysis again

$$(u \lor x \lor y) \land (x \lor \overline{y} \lor z) \land (\overline{x} \lor z) \land (\overline{y} \lor \overline{z}) \land (\overline{x} \lor \overline{z}) \land (\overline{u} \lor w) \land (\overline{u} \lor \overline{w})$$

Assignment “left on trail” always falsifies derived clause

$\Rightarrow$ every derived constraint “explains” conflict

Terminate conflict analysis when explanation looks nice

Learn asserting constraint: after backjump, some variable guaranteed to flip

Assignment "left on trail" always falsifies derived clause

$\Rightarrow$ every derived constraint “explains” conflict

Terminate conflict analysis when explanation looks nice

Learn asserting constraint: after backjump, some variable guaranteed to flip
Generalized Resolution

Can mimic resolution step

\[
\frac{x \lor \overline{y} \lor z \quad \overline{y} \lor \overline{z}}{x \lor \overline{y}}
\]
Generalized Resolution

Can mimic resolution step

\[
\frac{x \lor \overline{y} \lor z \quad \overline{y} \lor \overline{z}}{x \lor \overline{y}}
\]

by adding clauses as pseudo-Boolean constraints

\[
\begin{align*}
x + \overline{y} + z & \geq 1 \\
\overline{y} + \overline{z} & \geq 1 \\
x + 2\overline{y} & \geq 1
\end{align*}
\]

(Recall \(z + \overline{z} = 1\))
Generalized Resolution

Can mimic resolution step

$\frac{x \lor \overline{y} \lor z}{x \lor \overline{y}} \frac{\overline{y} \lor \overline{z}}{x \lor \overline{y}}$

by adding clauses as pseudo-Boolean constraints

$x + \overline{y} + z \geq 1 \quad \overline{y} + \overline{z} \geq 1 \quad \frac{x + 2\overline{y} \geq 1}{(\text{Recall } z + \overline{z} = 1)}$

Generalized resolution rule (from [Hoo88, Hoo92])

Positive linear combination so that some variable cancels

$\frac{a_1 x_1 + \sum_{i \geq 2} a_i \ell_i \geq A}{\sum_{i \geq 2} \left( \frac{c}{a_1} a_i + \frac{c}{b_1} b_i \right) \ell_i \geq \frac{c}{a_1} A + \frac{c}{b_1} B - c} \quad \frac{b_1 \overline{x}_1 + \sum_{i \geq 2} b_i \ell_i \geq B}{[c = \text{lcm}(a_1, b_1)]}$
Saturation

Actually, don’t get quite the right constraint in mimicking of resolution

\[
\begin{align*}
  x + \overline{y} + z & \geq 1 \\
  \overline{y} + \overline{z} & \geq 1 \\
  x + 2\overline{y} & \geq 1
\end{align*}
\]
Saturation

Actually, don’t get quite the right constraint in mimicking of resolution

\[
\begin{align*}
x + \overline{y} + z & \geq 1 \\
\overline{y} + \overline{z} & \geq 1 \\
x + 2\overline{y} & \geq 1
\end{align*}
\]

But clearly valid to conclude

\[
\begin{align*}
x + 2\overline{y} & \geq 1 \\
x + \overline{y} & \geq 1
\end{align*}
\]
Saturation

Actually, don’t get quite the right constraint in mimicking of resolution

\[
\begin{align*}
    x + \bar{y} + z & \geq 1 \\
    \bar{y} + \bar{z} & \geq 1 \\
    x + 2\bar{y} & \geq 1
\end{align*}
\]

But clearly valid to conclude

\[
\begin{align*}
    x + 2\bar{y} & \geq 1 \\
    x + \bar{y} & \geq 1
\end{align*}
\]

Saturation rule

\[
\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min\{a_i, A\} \cdot \ell_i \geq A}
\]

Sound over integers, not over rationals (need such rules for SAT solving)
Saturation

Actually, don’t get quite the right constraint in mimicking of resolution

\[
\begin{align*}
x + \bar{y} + z & \geq 1 \\
\bar{y} + z & \geq 1 \\
x + 2\bar{y} & \geq 1
\end{align*}
\]

But clearly valid to conclude

\[
\begin{align*}
x + 2\bar{y} & \geq 1 \\
x + \bar{y} & \geq 1
\end{align*}
\]

Saturation rule

\[
\frac{\sum_i a_i \ell_i \geq A}{\sum_i \min\{a_i, A\} \cdot \ell_i \geq A}
\]

Sound over integers, not over rationals (need such rules for SAT solving)

[Generalized resolution as defined in [Hoo88, Hoo92] includes fix above, but convenient here to make the two separate steps explicit]
Analyze Conflict with Generalized Resolution + Saturation!

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]
Analyze Conflict with Generalized Resolution + Saturation!

\[ C_1 \triangleq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \triangleq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \equiv 0, x_2 \overset{C_1}{\equiv} 1, x_3 \overset{C_1}{\equiv} 1 \} \Rightarrow \text{Conflict with } C_2 \)

(Note: same constraint can propagate several times!)
Analyze Conflict with Generalized Resolution + Saturation!

\[
C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4
\]

\[
C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3
\]

Trail \( \rho = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \) \Rightarrow Conflict with \( C_2 \)

(Note: same constraint can propagate several times!)

- Resolve \( reason(x_3, \rho) \doteq C_1 \) with \( C_2 \) over \( x_3 \) to get \( resolve(C_1, C_2, x_3) \)

\[
\begin{align*}
2x_1 + 2x_2 + 2x_3 + x_4 & \geq 4 \\
2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 & \geq 3 \\
x_4 & \geq 1
\end{align*}
\]
Analyze Conflict with Generalized Resolution + Saturation!

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow \text{Conflict with } C_2 \\
(\text{Note: same constraint can propagate several times!})

- Resolve \( \text{reason}(x_3, \rho) \doteq C_1 \) with \( C_2 \) over \( x_3 \) to get \( \text{resolve}(C_1, C_2, x_3) \)
  \[
  \begin{align*}
  2x_1 + 2x_2 + 2x_3 + x_4 & \geq 4 \\
  x_4 & \geq 1 \\
  2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 & \geq 3 
  \end{align*}
  \]

- Applying \( \text{saturate}(x_4 \geq 1) \) does nothing
Analyze Conflict with Generalized Resolution + Saturation!

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \Rightarrow \text{Conflict with } C_2 \)

(Note: same constraint can propagate several times!)

- Resolve \( \text{reason}(x_3, \rho) \doteq C_1 \) with \( C_2 \) over \( x_3 \) to get \( \text{resolve}(C_1, C_2, x_3) \)

\[
\begin{align*}
2x_1 + 2x_2 + 2x_3 + x_4 & \geq 4 \quad & 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 & \geq 3 \\
x_4 & \geq 1
\end{align*}
\]

- Applying \( \text{saturate}(x_4 \geq 1) \) does nothing
- Non-negative slack w.r.t. \( \rho' = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1 \} \) — not conflicting!
What Went Wrong? And What to Do About It?

**Accident report**

- Generalized resolution *sound over the reals*
- Given \( \rho' = \{ x_1 = 0, x_2 = 1 \} \), over the reals have
  - \( C_1 \trianglerighteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \) propagates \( x_3 \geq \frac{1}{2} \)
  - \( C_2 \trianglerighteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \) satisfied by \( x_3 \leq \frac{1}{2} \)
- So after resolving away \( x_3 \), “can’t see any conflict”
What Went Wrong? And What to Do About It?

**Accident report**
- Generalized resolution sound over the reals
- Given $\rho' = \{x_1 = 0, x_2 = 1\}$, over the reals have
  - $C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$ propagates $x_3 \geq \frac{1}{2}$
  - $C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$ satisfied by $x_3 \leq \frac{1}{2}$
- So after resolving away $x_3$, “can’t see any conflict”

**Remedial action**
- Strengthen propagation to $x_3 \geq 1$ also over the reals
- I.e., want reason $C$ with $slack(C; \rho') = 0$
- **Fix (non-obvious):** Apply weakening
  $$\text{weaken}(\sum_i a_i \ell_i \geq A, \ell_j) = \sum_{i \neq j} a_i \ell_i \geq A - a_j$$
  to reason constraint and then saturate
- Approach in [CK05] (seems to go back to observations in [Wil76])
Try to Reduce the Reason Constraint

\[ C_1 \equiv 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \equiv 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \) \( \Rightarrow \) Conflict with \( C_2 \)

Let’s try to

1. Weaken reason on non-falsified literal (but not last propagated)
2. Saturate weakened constraint
3. Resolve with conflicting constraint over propagated literal
Try to Reduce the Reason Constraint

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \Rightarrow \text{Conflict with } C_2 \)

Let’s try to:

1. Weaken reason on non-falsified literal (but not last propagated)
2. Saturate weakened constraint
3. Resolve with conflicting constraint over propagated literal

\[
\begin{align*}
\text{weaken } x_2 & \quad \frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 + x_4 \geq 2} \\
\text{saturate} & \quad \frac{2x_1 + 2x_3 + x_4 \geq 2}{2x_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3} \quad 2\overline{x}_2 + x_4 \geq 1
\end{align*}
\]
Try to Reduce the Reason Constraint

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \) \( \Rightarrow \) Conflict with \( C_2 \)

Let’s try to

1. Weaken reason on non-falsified literal (but not last propagated)
2. Saturate weakened constraint
3. Resolve with conflicting constraint over propagated literal

\[
\begin{align*}
\text{weaken } x_2 & \quad \frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 + x_4 \geq 2} \\
\text{saturate} & \quad \frac{2x_1 + 2x_3 + x_4 \geq 2}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3} \\
\text{resolve } x_3 & \quad \frac{2\overline{x}_2 + x_4 \geq 1}{2\overline{x}_2 + x_4 \geq 1}
\end{align*}
\]

Bummer! Still non-negative slack — not conflicting
Try Again to Reduce the Reason Constraint. . .

\[ C_1 \equiv 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \equiv 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \) \Rightarrow Conflict with \( C_2 \)
Try Again to Reduce the Reason Constraint. . .

\[ C_1 \equiv 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \equiv 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \Rightarrow \text{Conflict with } C_2 

\begin{align*}
\text{weaken } \{x_2, x_4\} & \quad \frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 \geq 1} \\
\text{saturate} & \quad \frac{2x_1 + 2x_3 \geq 1}{x_1 + x_3 \geq 1} \\
\text{resolve } x_3 & \quad \frac{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{2\overline{x}_2 \geq 1}
\end{align*}
Try Again to Reduce the Reason Constraint. . .

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \) \implies Conflict with \( C_2 \)

\[
\begin{align*}
\text{weaken } \{ x_2, x_4 \} & \quad 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \\
\text{saturate} & \quad 2x_1 + 2x_3 \geq 1 \\
\text{resolve } x_3 & \quad x_1 + x_3 \geq 1 \\
& \quad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \\
& \quad 2\overline{x}_2 \geq 1
\end{align*}
\]

Negative slack — conflicting!
Try Again to Reduce the Reason Constraint. . .

\[
C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \\
C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3
\]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \) \implies Conflict with \( C_2 \)

\[
\text{weaken } \{x_2, x_4\} \quad \frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_3 \geq 1} \\
\quad \frac{2x_1 + 2x_3 \geq 1}{x_1 + x_3 \geq 1} \\
\quad \frac{x_1 + x_3 \geq 1}{2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3} \\
\text{saturate} \quad \frac{2\overline{x}_2 \geq 1}{\text{resolve } x_3}
\]

Negative slack — conflicting!

Backjump propagates to conflict without decisions

Done! Next conflict analysis will derive contradiction

(Or, in practice, terminate immediately when conflict without decisions)
Reason Reduction Using Saturation [CK05]

\[
\text{reduceSat}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho) \]

\[\textbf{while } \text{slack}(\text{resolve}(C_{\text{confl}}, C_{\text{reason}}, \ell); \rho) \geq 0 \textbf{ do} \]
\[\quad \ell' \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ not falsified by } \rho; \]
\[\quad C_{\text{reason}} \leftarrow \text{saturate}(\text{weaken}(C_{\text{reason}}, \ell')); \]
\[\textbf{end} \]
\[\textbf{return } C_{\text{reason}}; \]

Why does this work?

Slack is subadditive:

\[
\text{slack}(c \cdot C + d \cdot D; \rho) \leq c \cdot \text{slack}(C; \rho) + d \cdot \text{slack}(D; \rho)
\]

By invariant have \(\text{slack}(C_{\text{confl}}; \rho) < 0\).

Weakening leaves \(\text{slack}(C_{\text{reason}}; \rho)\) unchanged.

Saturation decreases slack — reach 0 when max #literals weakened.
Reason Reduction Using Saturation [CK05]

\[ \text{reduceSat}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho) \]

\[
\text{while} \quad \text{slack}(\text{resolve}(C_{\text{confl}}, C_{\text{reason}}, \ell); \rho) \geq 0 \quad \text{do}
\]
\[
\quad \ell' \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ not falsified by } \rho;
\]
\[
\quad C_{\text{reason}} \leftarrow \text{saturate(weaken}(C_{\text{reason}}, \ell'));
\]
\[
\text{end}
\]
\[
\text{return } C_{\text{reason}};
\]

Why does this work?

- Slack is subadditive

\[
\text{slack}(c \cdot C + d \cdot D; \rho) \leq c \cdot \text{slack}(C; \rho) + d \cdot \text{slack}(D; \rho)
\]
Reason Reduction Using Saturation [CK05]

\[
\text{reduceSat}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho)
\]

\[
\begin{align*}
\text{while} \ & \ slack(\text{resolve}(C_{\text{confl}}, C_{\text{reason}}, \ell); \rho) \geq 0 \\
& \quad \ell' \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ not falsified by } \rho; \\
& \quad C_{\text{reason}} \leftarrow \text{saturate(weaken}(C_{\text{reason}}, \ell'));
\end{align*}
\]

\[
\text{return } C_{\text{reason}};
\]

Why does this work?

- Slack is subadditive

\[
slack(c \cdot C + d \cdot D; \rho) \leq c \cdot slack(C; \rho) + d \cdot slack(D; \rho)
\]

- By invariant have \( slack(C_{\text{confl}}; \rho) < 0 \)
reduceSat\((C_{\text{confl}}, C_{\text{reason}}, \ell, \rho)\)

\[
\text{while } \text{slack}(\text{resolve}(C_{\text{confl}}, C_{\text{reason}}, \ell); \rho) \geq 0 \text{ do}
\quad \ell' \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ not falsified by } \rho;
\quad C_{\text{reason}} \leftarrow \text{saturate}\left(\text{weaken}(C_{\text{reason}}, \ell')\right);
\text{end}
\]
\text{return } C_{\text{reason}};

Why does this work?

- Slack is subadditive

\[
\text{slack}(c \cdot C + d \cdot D; \rho) \leq c \cdot \text{slack}(C; \rho) + d \cdot \text{slack}(D; \rho)
\]

- By invariant have \(\text{slack}(C_{\text{confl}}; \rho) < 0\)

- **Weakening leaves** \(\text{slack}(C_{\text{reason}}; \rho)\) unchanged
Reason Reduction Using Saturation [CK05]

\[
\text{reduceSat}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho)
\]

\begin{algorithm}
\begin{algorithmic}
\While {slack(resolve($C_{\text{confl}}, C_{\text{reason}}, \ell; \rho$)) \geq 0}
\State $\ell' \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ not falsified by } \rho$
\State $C_{\text{reason}} \leftarrow \text{saturate(weaken($C_{\text{reason}}, \ell'$))}$
\EndWhile
\State \text{return } C_{\text{reason}};
\end{algorithmic}
\end{algorithm}

Why does this work?

- Slack is subadditive
  \[
  \text{slack}(c \cdot C + d \cdot D; \rho) \leq c \cdot \text{slack}(C; \rho) + d \cdot \text{slack}(D; \rho)
  \]
- By invariant have $\text{slack}(C_{\text{confl}}; \rho) < 0$
- Weakening leaves $\text{slack}(C_{\text{reason}}; \rho)$ unchanged
- Saturation decreases slack — reach 0 when max \# literals weakened
Pseudo-Boolean Conflict Analysis

\[
\text{analyzePBconflict}(C_{\text{confl}}, \rho)
\]

\[
\text{while } C_{\text{confl}} \text{ not asserting do}
\]

\[
\ell \leftarrow \text{literal assigned last on trail } \rho;
\]

\[
\text{if } \bar{\ell} \text{ occurs in } C_{\text{confl}} \text{ then}
\]

\[
C_{\text{reason}} \leftarrow \text{reason}(\ell, \rho);
\]

\[
C_{\text{reason}} \leftarrow \text{reduceSat}(C_{\text{reason}}, C_{\text{confl}}, \ell, \rho);
\]

\[
C_{\text{confl}} \leftarrow \text{resolve}(C_{\text{confl}}, C_{\text{reason}}, \ell);
\]

\[
C_{\text{confl}} \leftarrow \text{saturate}(C_{\text{confl}});
\]

\[
\rho \leftarrow \text{removeLast}(\rho);
\]

\[
\text{end}
\]

\[
\text{end}
\]

\[
\text{return } C_{\text{confl}};
\]

Reduction of reason \textit{new compared to CDCL} — everything else the same

Essentially conflict analysis used in \textit{SAT4J} [LP10]
Some Problems Compared to CDCL

- Compared to clauses **harder to detect propagation** for constraints like

$$\sum_{i=1}^{n} x_i \geq n - 1$$
Some Problems Compared to CDCL

- Compared to clauses **harder to detect propagation** for constraints like
  \[ \sum_{i=1}^{n} x_i \geq n - 1 \]

- Generalized resolution for general pseudo-Boolean constraints
  \[ \Rightarrow \text{lots of lcm computations} \]
  \[ \Rightarrow \text{coefficient sizes can explode} \text{ (expensive arithmetic)} \]
Some Problems Compared to CDCL

- Compared to clauses **harder to detect propagation** for constraints like
  \[ \sum_{i=1}^{n} x_i \geq n - 1 \]

- Generalized resolution for general pseudo-Boolean constraints
  ⇒ lots of \text{lcm} computations
  ⇒ **coefficient sizes can explode** (expensive arithmetic)

- For CNF inputs, **degenerates to resolution!**
  ⇒ CDCL but with super-expensive data structures
The Cutting Planes Proof System

Cutting planes as defined in theory literature [CCT87] doesn’t use saturation but instead division (a.k.a. Chvátal-Gomory cut)

**Literal axioms**

\[ \ell_i \geq 0 \]

**Linear combination**

\[ \sum_i a_i \ell_i \geq A \quad \sum_i b_i \ell_i \geq B \]
\[ \sum_i (c_A a_i + c_B b_i) \ell_i \geq c_A A + c_B B \]

**Division**

\[ \sum_i a_i \ell_i \geq A \]
\[ \sum_i \lceil a_i / c \rceil \ell_i \geq \lceil A / c \rceil \]
The Cutting Planes Proof System

Cutting planes as defined in theory literature [CCT87] doesn’t use saturation but instead division (a.k.a. Chvátal-Gomory cut)

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

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

**Division**  
\[ \sum_i a_i \ell_i \geq A \]
\[ \sum_i \left\lceil \frac{a_i}{c} \right\rceil \ell_i \geq \left\lceil \frac{A}{c} \right\rceil \]

- Cutting planes with division implicationally complete
- Cutting planes with saturation is not [VEG+18]
- Can division yield stronger conflict analysis?
The Cutting Planes Proof System

Cutting planes as defined in theory literature [CCT87] doesn’t use saturation but instead division (a.k.a. Chvátal-Gomory cut).

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

**Linear combination**
\[ \begin{align*}
\sum_i a_i l_i & \geq A \\
\sum_i b_i l_i & \geq B \\
\sum_i (c_A a_i + c_B b_i) l_i & \geq c_A A + c_B B
\end{align*} \]

**Division**
\[ \frac{\sum_i a_i l_i \geq A}{\sum_i \lceil a_i/c \rceil l_i \geq \lceil A/c \rceil} \]

- Cutting planes with division implicationally complete
- Cutting planes with saturation is not [VEG+18]
- Can division yield stronger conflict analysis? (Used for general integer linear programming in CUTSAT [JdM13])
Using Division to Reduce the Reason

\[ C_1 \equiv 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \equiv 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \overline{d} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \) \( \Rightarrow \) Conflict with \( C_2 \)
Using Division to Reduce the Reason

\[ C_1 \triangleq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \triangleq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \overset{d}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1 \} \) \Rightarrow \text{Conflict with } C_2

1. Weaken reason on non-falsified literal(s) with coefficient not divisible by propagating literal coefficient
2. Divide weakened constraint by propagating literal coefficient
3. Resolve with conflicting constraint over propagated literal
Using Division to Reduce the Reason

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \doteq 1, x_3 \doteq 1 \} \Rightarrow \text{Conflict with } C_2 \)

1. Weaken reason on non-falsified literal(s) with coefficient not divisible by propagating literal coefficient
2. Divide weakened constraint by propagating literal coefficient
3. Resolve with conflicting constraint over propagated literal

Weaken \( x_4 \)
\[
\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4}{2x_1 + 2x_2 + 2x_3 \geq 3}
\]
Divide by 2
\[
\frac{x_1 + x_2 + x_3 \geq 2}{0 \geq 1}
\]
Resolve \( x_3 \)
Using Division to Reduce the Reason

\[ C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \]
\[ C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \]

Trail \( \rho = \{ x_1 \doteq 0, x_2 \overset{C_1}{\doteq} 1, x_3 \overset{C_1}{\doteq} 1 \} \Rightarrow \text{Conflict with } C_2 \)

1. Weaken reason on non-falsified literal(s) with coefficient not divisible by propagating literal coefficient
2. Divide weakened constraint by propagating literal coefficient
3. Resolve with conflicting constraint over propagated literal

\[
\begin{align*}
\text{weaken } x_4 & \quad 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \\
\text{divide by 2} & \quad 2x_1 + 2x_2 + 2x_3 \geq 3 \\
\text{resolve } x_3 & \quad x_1 + x_2 + x_3 \geq 2 \\
& \quad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3 \\
& \quad 0 \geq 1
\end{align*}
\]

Terminate immediately!
Reason Reduction Using Division [EN18]

\[
\text{reduceDiv}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho) \\
\]
\[
c ← \text{coeff}(C_{\text{reason}}, \ell); \\
\text{while slack(resolve}(C_{\text{confl}}, \text{divide}(C_{\text{reason}}, c), \ell); \rho) \geq 0 \text{ do} \\
\quad \ell_j ← \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ such that } \ell_j \notin \rho \text{ and } c \nmid \text{coeff}(C, \ell_j); \\
\quad C_{\text{reason}} ← \text{weaken}(C_{\text{reason}}, \ell_j); \\
\text{end} \\
\text{return divide}(C_{\text{reason}}, c); \\
\]
reduceDiv\((C_{\text{confl}}, C_{\text{reason}}, \ell, \rho)\)

\[
c \leftarrow \text{coeff}(C_{\text{reason}}, \ell);
\]

\[
\text{while} \quad \text{slack}(\text{resolve}(C_{\text{confl}}, \text{divide}(C_{\text{reason}}, c, \ell); \rho)) \geq 0 \quad \text{do}
\]

\[
\ell_j \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ such that } \ell_j \notin \rho \text{ and } c \nmid \text{coeff}(C, \ell_j);
\]

\[
C_{\text{reason}} \leftarrow \text{weaken}(C_{\text{reason}}, \ell_j);
\]

\[
\text{end}
\]

\[
\text{return } \text{divide}(C_{\text{reason}}, c);
\]

So now why does this work?

- Sufficient to get reason with slack 0 since
  - 1. \(\text{slack}(C_{\text{confl}}; \rho) < 0\)
  - 2. slack is subadditive
Reason Reduction Using Division [EN18]

\[ \text{reduceDiv}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho) \]

\[ c \leftarrow \text{coeff}(C_{\text{reason}}, \ell); \]
\[ \textbf{while} \; \text{slack}(\text{resolve}(C_{\text{confl}}, \text{divide}(C_{\text{reason}}, c, \ell); \rho) \geq 0 \; \textbf{do} \]
\[ \quad \ell_j \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ such that } \ell_j \notin \rho \text{ and } c \nmid \text{coeff}(C, \ell_j); \]
\[ \quad C_{\text{reason}} \leftarrow \text{weaken}(C_{\text{reason}}, \ell_j); \]
\[ \textbf{end} \]
\[ \textbf{return} \; \text{divide}(C_{\text{reason}}, c); \]

So now why does this work?

- Sufficient to get reason with slack 0 since
  1. \[ \text{slack}(C_{\text{confl}}; \rho) < 0 \]
  2. slack is subadditive

- Weakening doesn’t change slack \( \Rightarrow \) always \( 0 \leq \text{slack}(C_{\text{reason}}; \rho) < c \)
Reason Reduction Using Division [EN18]

\begin{align*}
\text{reduceDiv}(C_{\text{confl}}, C_{\text{reason}}, \ell, \rho) & \\
\text{c} & \leftarrow \text{coeff}(C_{\text{reason}}, \ell); \\
\text{while} & \ \text{slack(resolve}(C_{\text{confl}}, \text{divide}(C_{\text{reason}}, c), \ell); \rho) \geq 0 \ \text{do} \\
& \quad \ell_j \leftarrow \text{literal in } C_{\text{reason}} \setminus \{\ell\} \text{ such that } \ell_j \notin \rho \text{ and } c \nmid \text{coeff}(C, \ell_j); \\
& \quad C_{\text{reason}} \leftarrow \text{weaken}(C_{\text{reason}}, \ell_j); \\
\text{end} \\
\text{return} & \ \text{divide}(C_{\text{reason}}, c); \\
\end{align*}

So now why does this work?

- Sufficient to get reason with slack 0 since
  - \(1\) \(\text{slack}(C_{\text{confl}}; \rho) < 0\)
  - \(2\) slack is subadditive
- Weakening doesn’t change slack \(\Rightarrow\) always \(0 \leq \text{slack}(C_{\text{reason}}; \rho) < c\)
- After max \#weakenings have \(0 \leq \text{slack}(\text{divide}(C_{\text{reason}}, c); \rho) < 1\)
Round-to-1 Reduction used in **ROUNDINGSat**

Reduction method used in **ROUNDINGSat** does max weakening right away

\[
\text{roundToOne}(C, \ell, \rho)
\]

\[
c \leftarrow \text{coeff}(C, \ell);
\]

\[
\text{foreach literal } \ell_j \text{ in } C \text{ do}
\]

\[
\quad \text{if } \bar{\ell}_j \notin \rho \text{ and } c \nmid \text{coeff}(C, \ell_j) \text{ then}
\]

\[
\quad \quad C \leftarrow \text{weaken}(C, \ell_j);
\]

\[
\quad \text{end}
\]

\[
\text{end}
\]

\[
\text{return divide}(C, c);
\]

And **roundToOne** used more aggressively in conflict analysis in [EN18] (though now we are dialling back on this...)

Jakob Nordström (UCPH & LU)
analyzePBconflict($C_{confl}, \rho$)

while $C_{confl}$ contains no or multiple falsified literals on last level do
  if no current solver decisions then
    output UNSATISFIABLE and terminate
  end
  $\ell \leftarrow$ literal assigned last on trail $\rho$;
  if $\bar{\ell}$ occurs in $C_{confl}$ then
    $C_{confl} \leftarrow$ roundToOne($C_{confl}, \bar{\ell}, \rho$);
    $C_{reason} \leftarrow$ roundToOne(reason($\ell, \rho$), $\ell, \rho$);
    $C_{confl} \leftarrow$ resolve($C_{confl}, C_{reason}, \ell$);
  end
  $\rho \leftarrow$ removeLast($\rho$);
end
$\ell \leftarrow$ literal in $C_{confl}$ last falsified by $\rho$;
return roundToOne($C_{confl}, \ell, \rho$);
Division vs. Saturation

- Higher conflict speed when PB reasoning doesn’t help [EN18]
- Seems to perform better when PB reasoning crucial [EGNV18]
- Keeps coefficients small — can (often) do fixed-precision arithmetic
- But SAT4J still better for some circuit verification problems [LBD+20]
- And still equally hard to detect propagation
- Also, still degenerates to resolution for CNF inputs
- Sometimes very poor performance even on infeasible 0-1 LPs!
Other PB Rules I: Cardinality Constraint Reduction

Given PB constraint

$$3x_1 + 2x_2 + x_3 + x_4 \geq 4$$

can compute least \#literals that have to be true
Other PB Rules I: Cardinality Constraint Reduction

Given PB constraint

\[ 3x_1 + 2x_2 + x_3 + x_4 \geq 4 \]

can compute least number of literals that have to be true

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]
Other PB Rules I: Cardinality Constraint Reduction

Given PB constraint

\[ 3x_1 + 2x_2 + x_3 + x_4 \geq 4 \]

can compute least \#literals that have to be true

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

**Galena** [CK05] only learns cardinality constraints — easier to deal with
Other PB Rules I: Cardinality Constraint Reduction

Given PB constraint

\[ 3x_1 + 2x_2 + x_3 + x_4 \geq 4 \]

can compute least \# literals that have to be true

\[ x_1 + x_2 + x_3 + x_4 \geq 2 \]

**Galena** [CK05] only learns cardinality constraints — easier to deal with

---

Cardinality constraint reduction rule

\[
\frac{\sum_i a_i l_i \geq A}{\sum_{i : a_i > 0} l_i \geq T} \quad T = \min\{|I| : I \subseteq [n], \sum_{i \in I} a_i \geq A\}
\]

Can be simulated with weakening + division
Other PB Rules II: Strengthening

Strengthening by example:

- Set $x = 0$ and propagate on constraints

\[
\begin{align*}
  x + y &\geq 1 \\
  x + z &\geq 1 \\
  y + z &\geq 1
\end{align*}
\]
Other PB Rules II: Strengthening

Strengthening by example:

- Set $x = 0$ and propagate on constraints

  $x + y \geq 1 \quad x + z \geq 1 \quad y + z \geq 1$

- $y \overset{x+y\geq1}{=} 1$ and $z \overset{x+z\geq1}{=} 1 \Rightarrow y + z \geq 1$ oversatisfied by margin 1
Other PB Rules II: Strengthening

Strengthening by example:

- Set \( x = 0 \) and propagate on constraints

\[
\begin{align*}
x + y \geq 1 \\
x + z \geq 1 \\
y + z \geq 1
\end{align*}
\]

- \( y \overset{x+y\geq1}{\Rightarrow} 1 \) and \( z \overset{x+z\geq1}{\Rightarrow} 1 \) \( \Rightarrow y + z \geq 1 \) oversatisfied by margin 1

- Hence, can deduce constraint \( x + y + z \geq 2 \)
Other PB Rules II: Strengthening

Strengthening by example:

- Set $x = 0$ and propagate on constraints
  
  \[
  x + y \geq 1 \quad x + z \geq 1 \quad y + z \geq 1
  \]

- $y \xRightarrow{x+y \geq 1} 1$ and $z \xRightarrow{x+z \geq 1} 1$ \implies $y + z \geq 1$ oversatisfied by margin 1

- Hence, can deduce constraint $x + y + z \geq 2$

**Strengthening rule** (imported by [DG02] from operations research)

- Suppose $\ell = 0 \implies \sum a_i \ell_i \geq A$ oversatisfied by amount $K$

- Then can deduce $K\ell + \sum a_i \ell_i \geq A + K$
Other PB Rules II: Strengthening

Strengthening by example:

- Set \( x = 0 \) and propagate on constraints
  \[
  x + y \geq 1 \quad x + z \geq 1 \quad y + z \geq 1
  \]

- \( y \overset{x+y\geq1} = 1 \) and \( z \overset{x+z\geq1} = 1 \) \( \Rightarrow y + z \geq 1 \) oversatisfied by margin 1

- Hence, can deduce constraint \( x + y + z \geq 2 \)

**Strengthening rule** (imported by [DG02] from operations research)

- Suppose \( \ell = 0 \) \( \Rightarrow \sum_i a_i \ell_i \geq A \) oversatisfied by amount \( K \)

- Then can deduce \( K\ell + \sum_i a_i \ell_i \geq A + K \)

In theory, can recover from bad encodings (e.g., CNF)
In practice, seems inefficient and hard to get to work...
Other PB Rules III: “Fusion Resolution”

Suppose have constraints

\[ 2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3 \]
Other PB Rules III: “Fusion Resolution”

Suppose have constraints
\[
2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3
\]
Then by eyeballing can conclude
\[
3y + 2z + w \geq 3
\]
Other PB Rules III: “Fusion Resolution”

Suppose have constraints

\[ 2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3 \]

Then by eyeballing can conclude

\[ 3y + 2z + w \geq 3 \]

But only get from resolution

\[ 6y + 4z + 2w \geq 4 \]
Other PB Rules III: “Fusion Resolution”

Suppose have constraints

\[ 2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3 \]

Then by eyeballing can conclude

\[ 3y + 2z + w \geq 3 \]

But only get from resolution + saturation

\[ 4y + 4z + 2w \geq 4 \]
Other PB Rules III: “Fusion Resolution”

Suppose have constraints

\[ 2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3 \]

Then by eyeballing can conclude

\[ 3y + 2z + w \geq 3 \]

But only get from resolution + saturation + division

\[ 2y + 2z + w \geq 2 \]
Other PB Rules III: “Fusion Resolution”

Suppose have constraints

\[ 2x + 3y + 2z + w \geq 3 \quad 2\overline{x} + 3y + 2z + w \geq 3 \]

Then by eyeballing can conclude

\[ 3y + 2z + w \geq 3 \]

But only get from resolution + saturation + division

\[ 2y + 2z + w \geq 2 \]

“Fusion resolution” [Goc17]

\[ a\ell + \sum_i b_i \ell_i \geq B \quad a\overline{\ell} + \sum_i b_i \ell_i \geq B' \]

\[ \sum_i b_i \ell_i \geq \min\{B, B'\} \]

No obvious way for cutting planes to immediately derive this
Shows up in some tricky benchmarks in [EGNV18]
Some PB Solving Challenges I: Input Format

1. **Preprocessing/presolving:** Important in SAT solving and integer linear programming, but not done in PB solvers — why?
   - Follow up on preliminary work on PB preprocessing in [MLM09]?
   - Use presolver PAPILO [PaP] from MIP solver SCIP [SCI]?

2. **CNF:** How to go beyond conflict-driven clause learning CDCL for decision problems encoded in CNF?

3. **Cardinality constraint detection:** Proposed as preprocessing [BLLM14] or inprocessing [EN20] — not yet competitive in practice

4. **Robustness:** Make PB solvers less sensitive to presence of extra constraints (anecdotally, CDCL solvers seem more stable)
Some PB Solving Challenges II: Conflict Analysis

1. **Choice of Boolean rule:**
   - Division, saturation, or select adaptively?
   - Or some other cut rule from ILP?
   - Try to avoid irrelevant literals? [LMMW20]

2. **Many more degrees of freedom** than in CDCL:
   - Skip resolution steps when slack very negative?
   - How aggressively to weaken reason in reduction step? [LMW20]
   - Learn general PB constraints or more limited form?
   - How far to backjump when learned constraint asserting at many levels?
   - How large precision to use in integer arithmetic?

3. **Do constraint minimization à la** [SB09, HS09]?

4. **How to assess quality of learned constraints?**

5. **Theoretical potential and limitations** poorly understood [VEG+18]
   - Separations of subsystems of cutting planes?
   - In particular, is division reasoning stronger than saturation? [GNY19]
Some PB Solving Challenges III: Solver Heuristics

Many heuristics more or less copied from CDCL — maybe tailor more carefully to PB setting?

1. **Variable selection**: VSIDS [MMZ+01] or VMTF [Rya04] or something else?

2. **Variable bumping**: Consider different bumping score depending on whether literal falsified, whether literal cancels, coefficient of literal and/or degree of constraint?

3. **Phase saving**: Standard as in [PD07], multiple phases [BF20], or something else?

4. **Different “modes”**: for SAT-focused and UNSAT-focused search?

See [Wal20] for a first in-depth investigation of some of these questions.
Some PB Solving Challenges IV: Efficiency and Correctness

1. Efficient **unit propagation** for PB constraints is a major challenge — latest news in [Dev20], but still much left to do

2. Efficient **detection of assertiveness** during conflict analysis

3. Efficient and concise **proof logging** for pseudo-Boolean solving (shameless self-plug: ongoing work on PB proof checker \texttt{VeriPB} [Ver19, GMN20b] in [EGMN20, GMN20a, GMM$^+$20, GN21])
Organization of This Tutorial

Part I: Pseudo-Boolean Preliminaries

Part II: Pseudo-Boolean Solving

Part III: Pseudo-Boolean Optimization

Part IV: Mixed Integer Linear Programming
Outline of Part III: Pseudo-Boolean Optimization

7 MaxSAT

8 Linear Search SAT-UNSAT (LSU)

9 Core-Guided Search

10 Implicit Hitting Set (IHS) Algorithm
MaxSAT Problem

Pseudo-Boolean optimization and MaxSAT solving intimately connected, so let’s do a detour and define MaxSAT

Weighted partial MaxSAT problem

**Input:** Soft clauses $C_1, \ldots, C_m$ with weights $w_i \in \mathbb{R}^+$, $i \in [m]$  
Hard clauses $C_{m+1}, \ldots, C_M$

**Goal:** Find assignment $\rho$ such that
- for all hard clauses $C_{m+1}, \ldots, C_M$ it holds that $\rho(C_j) = 1$
- $\rho$ maximizes $\sum_{i=1}^{m} \rho(C_i) w_i$

- All hard clauses must be satisfied
- Maximize weight of satisfied soft clauses = Minimize penalty of falsified soft clauses
- Write $(C)_w$ for clause $C$ with weight $w$ ($w = \infty$ for hard clause)
From MaxSAT to Pseudo-Boolean Optimization

MaxSAT instance

\[
\begin{align*}
(\overline{x})_5 \\
(y \lor \overline{z})_4 \\
(\overline{y} \lor z)_3 \\
(x \lor y \lor z)_\infty \\
(x \lor \overline{y} \lor \overline{z})_\infty
\end{align*}
\]
From MaxSAT to Pseudo-Boolean Optimization

MaxSAT instance

\[
\begin{align*}
(x)_5 \\
(y \lor \overline{z})_4 \\
(\overline{y} \lor z)_3 \\
(x \lor y \lor z)_\infty \\
(x \lor \overline{y} \lor \overline{z})_\infty
\end{align*}
\]

PBO instance

\[
\begin{align*}
\text{min} & \quad 5w_1 + 4w_2 + 3w_3 \\
& \quad w_1 + \overline{x} \geq 1 \\
& \quad w_2 + y + \overline{z} \geq 1 \\
& \quad w_3 + \overline{y} + z \geq 1 \\
& \quad x + y + z \geq 1 \\
& \quad x + \overline{y} + \overline{z} \geq 1
\end{align*}
\]
MaxSAT instance

\( (\overline{x})_5 \)
\( (y \lor \overline{z})_4 \)
\( (y \lor z)_3 \)
\( (x \lor y \lor z)_\infty \)
\( (x \lor \overline{y} \lor \overline{z})_\infty \)

PBO instance

\[
\begin{align*}
\text{min} & \quad 5w_1 + 4w_2 + 3w_3 \\
& \quad w_1 + \overline{x} \geq 1 \\
& \quad w_2 + y + \overline{z} \geq 1 \\
& \quad w_3 + \overline{y} + z \geq 1 \\
& \quad x + y + z \geq 1 \\
& \quad x + \overline{y} + \overline{z} \geq 1
\end{align*}
\]

So-called blocking variable transformation

Variables \( w_i \) are blocking or relaxation variables
MaxSAT instance

\[(\overline{x})_5\]
\[(y \lor \overline{z})_4\]
\[(\overline{y} \lor z)_3\]
\[(x \lor y \lor z)_\infty\]
\[(x \lor \overline{y} \lor \overline{z})_\infty\]

PBO instance

\[
\min 5w_1 + 4w_2 + 3w_3
\]
\[
w_1 + \overline{x} \geq 1
\]
\[
w_2 + y + \overline{z} \geq 1
\]
\[
w_3 + \overline{y} + z \geq 1
\]
\[
x + y + z \geq 1
\]
\[
x + \overline{y} + \overline{z} \geq 1
\]

So-called **blocking variable transformation**

Variables \(w_i\) are **blocking** or **relaxation** variables

Optimal solution \(\rho = \{ x = 0, y = 1, z = 0 \}\) with **penalty** 3
“MaxSAT instance” but with PB constraints:

Weighted Boolean Optimization [MMP09]
“MaxSAT instance” but with PB constraints:

Weighted Boolean Optimization [MMP09]

**PBO instance**

\[
\min \sum_{i=1}^{n} a_i w_i \\
C_1 \\
C_2 \\
\vdots \\
C_M
\]
From Pseudo-Boolean Optimization to MaxSAT/WBO

“MaxSAT instance” but with PB constraints:

Weighted Boolean Optimization [MMP09]

<table>
<thead>
<tr>
<th>PBO instance</th>
<th>MaxSAT/WBO instance</th>
</tr>
</thead>
</table>
| \[
\min \sum_{i=1}^{n} a_i w_i \\
C_1 \\
C_2 \\
\vdots \\
C_M
\] | \[
(\overline{w}_1)a_1 \\
\vdots \\
(\overline{w}_n)a_n \\
(C_1)_\infty \\
\vdots \\
(C_M)_\infty
\] |
Flavours of MaxSAT

- **Partial MaxSAT**: Hard and soft clauses
- **MaxSAT**: Only soft clauses
- **Unweighted MaxSAT**: All soft clauses have same weight (w.l.o.g. 1)
- **Weighted MaxSAT**: Different weights for soft clauses

4 different subproblems
But most current solvers deal with the most general problem
Main Approaches for MaxSAT Solving (and PBO)

1. Linear search SAT-UNSAT (LSU) (or model-improving search)
2. Core-guided search
3. Implicit hitting set (IHS) algorithm
Main Approaches for MaxSAT Solving (and PBO)

1. Linear search SAT-UNSAT (LSU) (or model-improving search)
2. Core-guided search
3. Implicit hitting set (IHS) algorithm

Will describe all of these algorithms as trying to

- minimize $\sum_{i=1}^{n} a_i w_i$
- subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$
  (possibly clausal)
Linear Search SAT-UNSAT (LSU) Algorithm

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Set $\rho_{\text{best}} = \emptyset$ and repeat the following:

1. Run SAT/PB solver
2. If solver returns UNSATISFIABLE, output $\rho_{\text{best}}$ and terminate
3. Otherwise, let $\rho_{\text{best}} :=$ returned solution $\rho$
4. Add constraint $\sum_{i=1}^{n} a_i w_i \leq -1 + \sum_{i=1}^{n} a_i \cdot \rho(w_i)$
5. Start over from the top
Linear Search Toy Example

1. Given PB formula $F$ and objective function
   $\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$
Given PB formula $F$ and objective function
\[
\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6
\]
Solver run on $F$ returns $\rho_1 = \{w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1\}$
Linear Search Toy Example

1. Given PB formula $F$ and objective function
   \[ \min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \]

2. Solver run on $F$ returns $\rho_1 = \{ w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1 \}$

3. Yields objective value $0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 9$, so add

   \[ w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 8 \]
**Linear Search Toy Example**

1. Given PB formula $F$ and objective function
   $$\text{min } w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$
2. Solver run on $F$ returns $\rho_1 = \{ w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1 \}$
3. Yields objective value $0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 9$, so add
   $$w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 8$$
4. Solver run on $F$ plus this new constraint returns $\rho_2 = \{ w_1 = w_3 = w_5 = w_6 = 0; w_2 = w_4 = 1 \}$
Linear Search Toy Example

1. Given PB formula $F$ and objective function
   \[
   \min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6
   \]

2. Solver run on $F$ returns $\rho_1 = \{w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1\}$

3. Yields objective value $0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 9$, so add
   \[
   w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 8
   \]

4. Solver run on $F$ plus this new constraint returns $\rho_2 = \{w_1 = w_3 = w_5 = w_6 = 0; w_2 = w_4 = 1\}$

5. Yields objective value 6, so add
   \[
   w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 5
   \]
Linear Search Toy Example

1. Given PB formula $F$ and objective function
   \[ \min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \]

2. Solver run on $F$ returns $\rho_1 = \{ w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1 \}$

3. Yields objective value $0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 9$, so add
   \[ w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 8 \]

4. Solver run on $F$ plus this new constraint returns $\rho_2 = \{ w_1 = w_3 = w_5 = w_6 = 0; w_2 = w_4 = 1 \}$

5. Yields objective value $6$, so add
   \[ w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 5 \]

6. Now solver returns UNSATISFIABLE
Linear Search Toy Example

1. Given PB formula \( F \) and objective function
   \[
   \min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6
   \]

2. Solver run on \( F \) returns \( \rho_1 = \{w_1 = w_2 = w_3 = w_6 = 0; w_4 = w_5 = 1\} \)

3. Yields objective value \( 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 9 \), so add
   \[
   w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 8
   \]

4. Solver run on \( F \) plus this new constraint returns
   \( \rho_2 = \{w_1 = w_3 = w_5 = w_6 = 0; w_2 = w_4 = 1\} \)

5. Yields objective value 6, so add
   \[
   w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6 \leq 5
   \]

6. Now solver returns **UNSATISFIABLE**

7. Hence, minimum value of objective function subject to \( F \) is 6
Linear vs. Binary Search?

What if we run binary search instead of linear search? Conventional wisdom appears to be that linear search is better.
Linear vs. Binary Search?

What if we run binary search instead of linear search? Conventional wisdom appears to be that linear search is better.

Two possible explanations:

1. In theory, objective value could decrease by just 1 every time — in practice, tend to get much larger jumps.
2. Potentially very different cost for:
   - SAT calls (feasible instances where solver will find solution)
   - UNSAT calls (where solver determines no solution exists)
Linear vs. Binary Search?

What if we run binary search instead of linear search? Conventional wisdom appears to be that linear search is better.

Two possible explanations:

1. In theory, objective value could decrease by just 1 every time — in practice, tend to get much larger jumps.
2. Potentially very different cost for:
   - SAT calls (feasible instances where solver will find solution)
   - UNSAT calls (where solver determines no solution exists)

Properties of linear search SAT-UNSAT:

- Can get some decent solution quickly, even if not optimal one.
- Important for anytime solving (when time is limited and something is better than nothing).
- But get no estimate of how good the solution is.
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{best} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{\text{best}} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{\text{best}}$ and terminate
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{\text{best}} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{\text{best}}$ and terminate
3. Otherwise learn clause over assumption variables, say $w_1 \lor \cdots \lor w_k$
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{\text{best}} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{\text{best}}$ and terminate
3. Otherwise learn clause over assumption variables, say $w_1 \lor \cdots \lor w_k$
4. Means that soft clauses $\mathcal{K} = \{C_1, \ldots, C_k\}$ form a core — can’t satisfy $\mathcal{K}$ and all hard constraints
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{best} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{best}$ and terminate
3. Otherwise learn clause over assumption variables, say $w_1 \lor \cdots \lor w_k$
4. Means that soft clauses $\mathcal{K} = \{C_1, \ldots, C_k\}$ form a core — can’t satisfy $\mathcal{K}$ and all hard constraints
5. Introduce new variables $z_j \iff \sum_{i=1}^{k} w_i \geq j$
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{best} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{best}$ and terminate
3. Otherwise learn clause over assumption variables, say $w_1 \lor \cdots \lor w_k$
4. Means that soft clauses $\mathcal{K} = \{C_1, \ldots, C_k\}$ form a core — can’t satisfy $\mathcal{K}$ and all hard constraints
5. Introduce new variables $z_j \Leftrightarrow \sum_{i=1}^{k} w_i \geq j$
6. Update objective function and $val_{best}$ using $\sum_{i=1}^{k} w_i = 1 + \sum_{j=2}^{k} z_j$ to cancel at least one variable $w_i$
Core-Guided Search

- Minimize $\sum_{i=1}^{n} a_i w_i$
- Subject to collection of PB constraints $F = C_1 \land \cdots \land C_m$

Think first of this as MaxSAT instance with $w_i$ as blocking variables

Set $val_{\text{best}} = 0$ and repeat the following:

1. Run SAT solver with assumptions (pre-made decisions) $w_i = 0$ for all $w_i$ in objective function
2. If solver returns SATISFIABLE, output $val_{\text{best}}$ and terminate
3. Otherwise learn clause over assumption variables, say $w_1 \lor \cdots \lor w_k$
4. Means that soft clauses $\mathcal{K} = \{C_1, \ldots, C_k\}$ form a core — can’t satisfy $\mathcal{K}$ and all hard constraints
5. Introduce new variables $z_j \iff \sum_{i=1}^{k} w_i \geq j$
6. Update objective function and $val_{\text{best}}$ using $\sum_{i=1}^{k} w_i = 1 + \sum_{j=2}^{k} z_j$ to cancel at least one variable $w_i$
7. Start over from top with updated objective function
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
- Updating objective with new variables: OLL algorithm — used in
  - answer set programming [AKMS12]
  - MaxSAT solving [MDM14]
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
- Updating objective with new variables: OLL algorithm — used in
  - answer set programming [AKMS12]
  - MaxSAT solving [MDM14]
- In general pseudo-Boolean setting, no need to think of $w_i$ as markers for soft clauses — they are just literals in objective function
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL$^+$13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
- Updating objective with new variables: OLL algorithm — used in
  - answer set programming [AKMS12]
  - MaxSAT solving [MDM14]
- In general pseudo-Boolean setting, no need to think of $w_i$ as markers for soft clauses — they are just literals in objective function
- And rewriting very convenient — just use PB constraints without re-encoding
Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
- Updating objective with new variables: OLL algorithm — used in
  - answer set programming [AKMS12]
  - MaxSAT solving [MDM14]
- In general pseudo-Boolean setting, no need to think of $w_i$ as markers for soft clauses — they are just literals in objective function
- And rewriting very convenient — just use PB constraints without re-encoding
- Core-guided PB search: assume optimistically that objective can reach best imaginable value; derive contradiction if not possible
Core-Guided Search

Core-Guided Search for Pseudo-Boolean Optimization

- Original core-guided idea from [FM06]; see [MHL+13] for survey
- Core-guided search is kind of UNSAT-SAT linear search
- Updating objective with new variables: **OLL algorithm** — used in
  - answer set programming [AKMS12]
  - MaxSAT solving [MDM14]
- In general pseudo-Boolean setting, no need to think of \( w_i \) as markers for soft clauses — they are just literals in objective function
- And rewriting very convenient — just use PB constraints without re-encoding
- **Core-guided PB search**: assume optimistically that objective can reach best imaginable value; derive contradiction if not possible
- Let us try to explain by concrete example
Given same PB formula $F$ and objective function

$$\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$ (1)
Core-Guided Search Toy Example (1/3)

1. Given same PB formula $F$ and objective function

$$\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$

2. Run solver on $F$ with assumptions $w_i = 0$, $i \in [6]$
Core-Guided Search Toy Example (1/3)

1. Given same PB formula $F$ and objective function

$$\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$  \hspace{1cm} (1)

2. Run solver on $F$ with assumptions $w_i = 0, i \in [6]$

3. Suppose solver returns PB core constraint

$$3w_2 + 2w_3 + w_4 + w_5 \geq 4$$  \hspace{1cm} (2)
Core-Guided Search Toy Example (1/3)

1. Given same PB formula $F$ and objective function

$$\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$  \hspace{1cm} (1)

2. Run solver on $F$ with assumptions $w_i = 0$, $i \in [6]$

3. Suppose solver returns PB core constraint

$$3w_2 + 2w_3 + w_4 + w_5 \geq 4$$  \hspace{1cm} (2)

4. Round to nicer-to-work-with cardinality core constraint

$$w_2 + w_3 + w_4 + w_5 \geq 2$$  \hspace{1cm} (3)
Core-Guided Search Toy Example (1/3)

1. Given same PB formula $F$ and objective function

$$\min w_1 + 2w_2 + 3w_3 + 4w_4 + 5w_5 + 6w_6$$

(1)

2. Run solver on $F$ with assumptions $w_i = 0$, $i \in [6]$

3. Suppose solver returns PB core constraint

$$3w_2 + 2w_3 + w_4 + w_5 \geq 4$$

(2)

4. Round to nicer-to-work-with cardinality core constraint

$$w_2 + w_3 + w_4 + w_5 \geq 2$$

(3)

5. Introduce new, fresh variables $y_3$ and $y_4$ and constraints

$$w_2 + w_3 + w_4 + w_5 = 2 + y_3 + y_4$$

(4a)

$$y_3 \geq y_4$$

(4b)

to enforce that $y_j$ means "$w_2 + w_3 + w_4 + w_5 \geq j$"
Core-Guided Search Toy Example (2/3)

6. Multiply (4a) by 2 and add to (1) to cancel $w_2$ and get updated, equivalent objective function

$$w_1 + w_3 + 2w_4 + 3w_5 + 6w_6 + 2y_3 + 2y_4 + 4$$

and update $val_{\text{best}} = 4$
Multiply (4a) by 2 and add to (1) to cancel \(w_2\) and get updated, equivalent objective function

\[
w_1 + w_3 + 2w_4 + 3w_5 + 6w_6 + 2y_3 + 2y_4 + 4
\]

and update \(val_{\text{best}} = 4\)

Run solver on \(F\) assuming all literals in (5) being 0
Multiply (4a) by 2 and add to (1) to cancel \(w_2\) and get updated, equivalent objective function

\[
w_1 + w_3 + 2w_4 + 3w_5 + 6w_6 + 2y_3 + 2y_4 + 4
\]  
and update \(\text{val}_{\text{best}} = 4\)

Run solver on \(F\) assuming all literals in (5) being 0

Suppose solver returns the clausal core constraint

\[
w_4 + w_5 + w_6 + y_3 \geq 1
\]
Multiply (4a) by 2 and add to (1) to cancel \( w_2 \) and get updated, equivalent objective function

\[
w_1 + w_3 + 2w_4 + 3w_5 + 6w_6 + 2y_3 + 2y_4 + 4 \tag{5}
\]

and update \( \text{val}_{\text{best}} = 4 \)

Run solver on \( F \) assuming all literals in (5) being 0

Suppose solver returns the clausal core constraint

\[
w_4 + w_5 + w_6 + y_3 \geq 1 \tag{6}
\]

Introduce new variables \( z_2, z_3, z_4 \) and the constraints

\[
w_4 + w_5 + w_6 + y_3 = 1 + z_2 + z_3 + z_4 \tag{7a}
\]

\[
z_2 \geq z_3 \tag{7b}
\]

\[
z_3 \geq z_4 \tag{7c}
\]

to enforce that \( z_j \) means “\( w_4 + w_5 + w_6 + y_3 \geq j \)”
Multiply (7a) by 2 and add to (5) to get 3rd equivalent objective

\[ w_1 + w_3 + w_5 + 4w_6 + 2y_4 + 2z_2 + 2z_3 + 2z_4 + 6 \]  

and update \( \text{val}_{\text{best}} = 6 \)
Multiply (7a) by 2 and add to (5) to get 3rd equivalent objective

\[ w_1 + w_3 + w_5 + 4w_6 + 2y_4 + 2z_2 + 2z_3 + 2z_4 + 6 \]  

\text{and update } \text{val}_{\text{best}} = 6

For 3rd time run solver on $F$, assuming all literals in (8) being 0
Multiply (7a) by 2 and add to (5) to get 3rd equivalent objective

\[ w_1 + w_3 + w_5 + 4w_6 + 2y_4 + 2z_2 + 2z_3 + 2z_4 + 6 \]  

and update \( val_{\text{best}} = 6 \)

For 3rd time run solver on \( F \), assuming all literals in (8) being 0

Suppose solver reports it is possible to achieve

\[ \rho = \{ w_1 = w_3 = w_5 = w_6 = y_4 = z_2 = z_3 = z_4 = 0 \} \]
Multiply (7a) by 2 and add to (5) to get 3rd equivalent objective

\[ w_1 + w_3 + w_5 + 4w_6 + 2y_4 + 2z_2 + 2z_3 + 2z_4 + 6 \quad (8) \]

and update \( \text{val}_{\text{best}} = 6 \)

For 3rd time run solver on \( F \), assuming all literals in (8) being 0

Suppose solver reports it is possible to achieve

\[ \rho = \{ w_1 = w_3 = w_5 = w_6 = y_4 = z_2 = z_3 = z_4 = 0 \} \quad (9) \]

Under assignment (9) the equality (4a) simplifies to

\[ w_2 + w_4 = 2 + y_3 \quad (10) \]

which can hold only if \( y_3 = 0 \) and \( w_2 = w_4 = 1 \), and this also satisfies (7a). Hence, have recovered optimal solution 6 (as before)
Properties of (Pure) Core-Guided Search

- Can get decent lower bounds on solution quickly
- Helps to cut off parts of search space that are “too good to be true”
- But find no actual solution until the final, optimal one
- Also, no estimate of how good the lower bound is
- Linear search much better at finding solutions — how to get the best of both worlds?
Core-Guided Search

Improvements of Core-Guided Search (1/2)

**Weight stratification** [ABGL12]

Set only literals with largest weight in objective to 0 ⇒

1. More compact core; or
2. Decent solution found early on
Improvements of Core-Guided Search (1/2)

**Weight stratification** [ABGL12]
Set only literals with largest weight in objective to 0 ⇒
1. More compact core; or
2. Decent solution found early on

**Independent cores** [BJ17]
If found core constraint over $w_1, w_2, \ldots, w_k$, remove these literals from assumptions and immediately run solver again with remaining assumptions
Improvements of Core-Guided Search (1/2)

**Weight stratification** [ABGL12]
Set only literals with largest weight in objective to 0 ⇒
1. More compact core; or
2. Decent solution found early on

**Independent cores** [BJ17]
If found core constraint over $w_1, w_2, \ldots, w_k$, remove these literals from assumptions and immediately run solver again with remaining assumptions

**Core boosting** [BDS19]
Start with core-guided search to get good lower bound estimate; then switch to linear search to find optimal solution
Improvements of Core-Guided Search (1/2)

**Weight stratification** [ABGL12]
Set only literals with largest weight in objective to 0 ⇒
- More compact core; or
- Decent solution found early on

**Independent cores** [BJ17]
If found core constraint over $w_1, w_2, \ldots, w_k$, remove these literals from assumptions and immediately run solver again with remaining assumptions

**Core boosting** [BDS19]
Start with core-guided search to get good lower bound estimate; then switch to linear search to find optimal solution

**Hybrid/interleaving search** [ADMR15]
Switch back and forth repeatedly between core-guided and linear search — cumbersome in CNF-based solver, but fairly cheap (and efficient) in native pseudo-Boolean solver [DGD⁺21]
**Core minimization**

In CDCL-based solver, try to get smaller core clauses. For PB solver, not so clear how to do this (constraint minimization also interesting problem in general for PB conflict analysis)
Core minimization
In CDCL-based solver, try to get smaller core clauses. For PB solver, not so clear how to do this (constraint minimization also interesting problem in general for PB conflict analysis)

Lazy variables [MJML14, DGD$^+$21]
For real-world instances, rewriting of objective function can introduce huge numbers of new variables, slowing down the solver — so don’t introduce all variables in one go but only lazily as needed
Core-Guided Search

Improvements of Core-Guided Search (2/2)

Core minimization
In CDCL-based solver, try to get smaller core clauses. For PB solver, not so clear how to do this (constraint minimization also interesting problem in general for PB conflict analysis)

Lazy variables [MJML14, DGDL\textsuperscript{+}21]
For real-world instances, rewriting of objective function can introduce huge numbers of new variables, slowing down the solver — so don’t introduce all variables in one go but only lazily as needed

Inference strength of core-guided search?

- Extension variables very strong in theory, but hard to use in practice
- Core-guided search provides principled way of introducing them
- Can we characterize the power of this method?
Evaluation of Core-Guided PB Solver in [DGD⁺21]

RoundingSat variants with core-guided (CG) and linear search (LSU)

#instances solved to optimality; highlighting 1st, 2nd, and 3rd best
# Evaluation of Core-Guided PB Solver in [DGD+21]

**RoundingSat** variants with core-guided (CG) and linear search (LSU) #instances solved to optimality; highlighting 1st, 2nd, and 3rd best

<table>
<thead>
<tr>
<th>Method</th>
<th>PB16opt (1600)</th>
<th>MIPopt (291)</th>
<th>KNAP (783)</th>
<th>CRAFT (985)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid (interleave CG &amp; LSU)</td>
<td>968</td>
<td>78</td>
<td>306</td>
<td>639</td>
</tr>
<tr>
<td>HybridCl (w/ clausal cores)</td>
<td>937</td>
<td>75</td>
<td>298</td>
<td>618</td>
</tr>
<tr>
<td>HybridNL (w/ non-lazy variables)</td>
<td>936</td>
<td>70</td>
<td>186</td>
<td>607</td>
</tr>
<tr>
<td>HybridClNL (w/ both)</td>
<td>917</td>
<td>67</td>
<td>203</td>
<td>612</td>
</tr>
<tr>
<td>RoundingSat (only LSU)</td>
<td>853</td>
<td>75</td>
<td>341</td>
<td>309</td>
</tr>
<tr>
<td>Coreguided (only CG)</td>
<td>911</td>
<td>61</td>
<td>43</td>
<td>595</td>
</tr>
<tr>
<td>Coreboosted (10% CG, then LSU)</td>
<td>959</td>
<td>80</td>
<td>344</td>
<td>580</td>
</tr>
<tr>
<td>Sat4J</td>
<td>773</td>
<td>61</td>
<td>373</td>
<td>105</td>
</tr>
<tr>
<td>NaPS</td>
<td>896</td>
<td>65</td>
<td>111</td>
<td>345</td>
</tr>
<tr>
<td>SCIP</td>
<td>1057</td>
<td>125</td>
<td>765</td>
<td>642</td>
</tr>
</tbody>
</table>
Evaluation of Core-Guided PB Solver in [DGD+21]

RoundingSat variants with core-guided (CG) and linear search (LSU)

#instances solved to optimality; highlighting 1st, 2nd, and 3rd best

<table>
<thead>
<tr>
<th></th>
<th>PB16opt (1600)</th>
<th>MIPopt (291)</th>
<th>KNAP (783)</th>
<th>CRAFT (985)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid (interleave CG &amp; LSU)</td>
<td>968</td>
<td>78</td>
<td>306</td>
<td>639</td>
</tr>
<tr>
<td>HybridCl (w/ clausal cores)</td>
<td>937</td>
<td>75</td>
<td>298</td>
<td>618</td>
</tr>
<tr>
<td>HybridNL (w/ non-lazy variables)</td>
<td>936</td>
<td>70</td>
<td>186</td>
<td>607</td>
</tr>
<tr>
<td>HybridClNL (w/ both)</td>
<td>917</td>
<td>67</td>
<td>203</td>
<td>612</td>
</tr>
<tr>
<td>RoundingSat (only LSU)</td>
<td>853</td>
<td>75</td>
<td>341</td>
<td>309</td>
</tr>
<tr>
<td>Coreguided (only CG)</td>
<td>911</td>
<td>61</td>
<td>43</td>
<td>595</td>
</tr>
<tr>
<td>Coreboosted (10% CG, then LSU)</td>
<td>959</td>
<td>80</td>
<td>344</td>
<td>580</td>
</tr>
<tr>
<td>Sat4j</td>
<td>773</td>
<td>61</td>
<td>373</td>
<td>105</td>
</tr>
<tr>
<td>NAPS</td>
<td>896</td>
<td>65</td>
<td>111</td>
<td>345</td>
</tr>
<tr>
<td>SCIP</td>
<td>1057</td>
<td>125</td>
<td>765</td>
<td>642</td>
</tr>
</tbody>
</table>

Significant improvement over PB state of the art, but MIP still better
Core-Guided Search

Core-Guided PB Solving for PB16 benchmarks [DGD+21]

Cumulative plot for solver performance on PB16 optimization benchmarks

Also including
- weight stratification (strat)
- independent cores (ind)
Implicit Hitting Set (IHS) Algorithm (1/2)

- Minimize \( \sum_{i=1}^{n} a_i w_i \)
- Subject to collection of PB constraints \( F = C_1 \land \cdots \land C_m \)
  
  (consider clausal constraints)

As in core-guided search, use solving with assumptions, but maintain collection \( \mathcal{K} \) of learned core clauses

\[
\begin{align*}
C_1 & \equiv w_{1,1} \lor w_{1,2} \lor \cdots \lor w_{1,k_s} \\
C_2 & \equiv w_{2,1} \lor w_{2,2} \lor \cdots \lor w_{2,k_s} \\
\vdots & \\
C_s & \equiv w_{s,1} \lor w_{s,2} \lor \cdots \lor w_{s,k_s}
\end{align*}
\]
Implicit Hitting Set (IHS) Algorithm (2/2)

Set $\mathcal{K} = \emptyset$ and repeat the following:

1. Compute minimum hitting set for $\mathcal{K}$, i.e., $W = \{w_i\}$ s.t.
   - $W \cap C \neq \emptyset$ for all $C \in \mathcal{K}$ ($W$ is hitting set)
   - $\sum_{w_i \in W} w_i$ minimal among $W$ with this property.

2. Run the solver with assumptions
   \[
   \{w_i = 1 \mid w_i \in W\} \cup \{w_j = 0 \mid w_j \notin W\}
   \]

3. If solver found solution, it must be optimal (since hitting set is optimal), so return solution with value $\sum_{w_i \in W} w_i$

4. Otherwise, solver returns new core $C_{s+1}$ — add it to $\mathcal{K}$ and start over from top
More About the Hitting Sets

- Minimality is actually not needed except in the very final step
- Save time by computing “decent” hitting sets earlier on in the search
- How to find hitting set?
- This is itself a pseudo-Boolean optimization problem
  [as discussed in Part I of tutorial]
  - Run MIP solver
  - Or PB solver
Implicit Hitting Set (IHS) Algorithm

Implicit Hitting Set vs. Core-Guided

- IHS and core-guided approaches for MaxSAT seem orthogonal [Bac21]
- For MaxSAT problems with many interchangeable soft clauses, core-guided seems better (i.e., when it is not important exactly which of these clauses end up in core)
- For MaxSAT problems with many distinct weights, IHS seems better

Relation between IHS and core-guided search?

Provide a more precise theoretical comparison of IHS and core-guided search with simulations and/or separations

(Some theoretical work on related problems in, e.g., [FMSV20, MIB+19])
Some More Open Questions

Combine IHS and core-guided search in MaxSAT solving?
Recent work on this in [BBP20]
Some More Open Questions

Combine IHS and core-guided search in MaxSAT solving?
Recent work on this in [BBP20]

Combine IHS with pseudo-Boolean optimization?
- In PB setting, cores will not be subsets of clauses but PB constraints $C_1, \ldots, C_s$ over objective function literals
- Hitting set $W$ is partial assignment guaranteed to satisfy all constraints $C_1, \ldots, C_s$
- Want to find minimum-cost set $W$ of literals (w.r.t. objective function) with this property
- Not implemented in native PB solvers (to best of my knowledge)
Organization of This Tutorial

Part I: Pseudo-Boolean Preliminaries

Part II: Pseudo-Boolean Solving

Part III: Pseudo-Boolean Optimization

Part IV: Mixed Integer Linear Programming
Outline of Part IV: Mixed Integer Linear Programming

11 MIP and ILP Solving
- MIP Preliminaries
- Branch-and-Bound and Branch-and-Cut
- Additional Techniques

12 Combining PB and MIP Techniques
- Some Challenges When Integrating PB and LP Solving
- A Proof-of-Concept Hybrid PB-LP Solver
- Evaluation and Conclusions
Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{N}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n + 1, \ldots, N$
Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{N}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n + 1, \ldots, N$

- Linear constraints
- Integer-valued variables
- Real-valued variables
- Linear objective function
Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{N}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n + 1, \ldots, N$

Linera constraints
- Integer-valued variables
- Real-valued variables
- Linear objective function

No real-valued variables:
- integer linear program (ILP)
- $0 \leq x_j \leq 1$ for all $j$: 0-1 ILP
- Vacuous objective $\sum_j 0 \cdot x_j$:
- decision problem
- But MIP best for optimization
Two Differences Compared to SAT/PB

Academia vs. industry

- Best solvers are commercial and closed-source
- E.g., CPLEX [CPL], GUROBI [Gur], and XPRESS [Xpr]
- Academic solvers like SCIP [SCI] are excellent but not as good
Two Differences Compared to SAT/PB

**Academia vs. industry**
- Best solvers are commercial and closed-source
- E.g., CPLEX [CPL], Gurobi [Gur], and Xpress [Xpr]
- Academic solvers like SCIP [SCI] are excellent but not as good

**Search vs. backtracking**
- SAT/PB: Fast decisions; careful, slow(er) conflict analysis
- MIP: Lots of time & effort on decisions; backtracking not so advanced
MIP Solving at a High Level

1. Preprocessing (called presolving)

2. Linear programming + branch-and-bound

3. Add cutting planes ruling out infeasible LP-solutions (branch-and-cut method going back to [Gom58])

4. Heuristics for quickly finding good feasible solutions
Linear Programming Relaxation

Linear Programming Relaxation (LPR)

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{N}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n+1, \ldots, N$

- Fast to solve (just linear programming)
- LP solution $x^*$ yields lower bound
- Or, if $x^*$ “accidentally” feasible, have optimal solution
- Use simplex algorithm — will have many LP calls for same problem with different variable bounds; need efficient hot restarts
LP-Based Branch-and-Bound

Branch-and-bound

Choose integer-valued $x_j$ and $B \in \mathbb{N}$

- Solve MIP plus constraint $x_j \geq B$
- Solve MIP plus constraint $x_j \leq B - 1$
LP-Based Branch-and-Bound

Branch-and-bound

Choose integer-valued $x_j$ and $B \in \mathbb{N}$

- Solve MIP plus constraint $x_j \geq B$
- Solve MIP plus constraint $x_j \leq B - 1$

Creates (growing) branch-and-bound tree of subproblems
Prune subproblem/node when

- LP is infeasible
- LP bound $> \text{incumbent}$ (current best solution)
LP-Based Branch-and-Bound

Branch-and-bound

Choose integer-valued $x_j$ and $B \in \mathbb{N}$

- Solve MIP plus constraint $x_j \geq B$
- Solve MIP plus constraint $x_j \leq B - 1$

Creates (growing) branch-and-bound tree of subproblems

Prune subproblem/node when

- LP is infeasible
- LP bound $>$ incumbent (current best solution)

Branch on

- Variables
- General linear constraints (powerful but difficult)

Corresponds to stabbing planes proof system [BFI+18]
Branch-and-Cut

General cutting plane method

1. Solve LP relaxation
2. If solution $x^*$ feasible for MIP $\Rightarrow$ found optimum
3. Otherwise generate and add constraint $\sum_j b_j x_j \leq B$ that is
   - valid for MIP
   - violated by LP solution $x^*$
4. Repeat from the top
Branch-and-Cut

General cutting plane method

1. Solve LP relaxation
2. If solution $x^*$ feasible for MIP $\implies$ found optimum
3. Otherwise generate and add constraint $\sum_j b_j x_j \leq B$ that is
   - valid for MIP
   - violated by LP solution $x^*$
4. Repeat from the top

PB solving rules division and saturation are examples of cut rules
Branch-and-Cut

General cutting plane method

1. Solve LP relaxation
2. If solution $x^*$ feasible for MIP $\Rightarrow$ found optimum
3. Otherwise generate and add constraint $\sum_j b_j x_j \leq B$ that is
   - valid for MIP
   - violated by LP solution $x^*$
4. Repeat from the top

PB solving rules division and saturation are examples of cut rules

Branch-and-cut

- Run branch-and-bound
- But in each subproblem, use cutting plane method to repeatedly
  - solve LP relaxation
  - add cut
Example Cut 1: Knapsack Cover Cut

Given constraint

\[ \sum_{j \in I} a_j x_j \leq A \]

for \( x_j \in \{0, 1\} \) and \( a_j, A \in \mathbb{N}^+ \)
Example Cut 1: Knapsack Cover Cut

Given constraint

$$\sum_{j \in I} a_j x_j \leq A$$

for $$x_j \in \{0, 1\}$$ and $$a_j, A \in \mathbb{N}^+$$

Find minimal cover $$C \subset I$$ such that

$$\sum_{j \in C} a_j > A$$

$$\sum_{j \in C \setminus \{i\}} a_j \leq A$$ for all $$i \in C$$
Example Cut 1: Knapsack Cover Cut

Given constraint

\[
\sum_{j \in I} a_j x_j \leq A
\]

for \(x_j \in \{0, 1\}\) and \(a_j, A \in \mathbb{N}^+\)

Find minimal cover \(C \subset I\) such that

\[
\sum_{j \in C} a_j > A
\]

\[
\sum_{j \in C \setminus \{i\}} a_j \leq A \quad \text{for all } i \in C
\]

Then can derive

\[
\sum_{j \in I} x_j \leq |C| - 1
\]
Example Cut 1: Knapsack Cover Cut

Given constraint
\[ \sum_{j \in I} a_j x_j \leq A \]

for \( x_j \in \{0, 1\} \) and \( a_j, A \in \mathbb{N}^+ \)

Find minimal cover \( C \subset I \) such that
\[ \sum_{j \in C} a_j > A \]
\[ \sum_{j \in C \setminus \{i\}} a_j \leq A \text{ for all } i \in C \]

Then can derive
\[ \sum_{j \in I} x_j \leq |C| - 1 \]

(In cutting planes, weaken & divide \( \sum_{j \in I} a_j \bar{x}_j \geq -A + \sum_{j \in I} a_j \) to get disjunctive clause \( \sum_{j \in I} x_j \geq 1 \))
Mixed integer rounding (MIR) cut [MW01] applied to (normalized) pseudo-Boolean constraint

\[ \sum_i a_i \ell_i \geq A \]

with divisor \( d \in \mathbb{N}^+ \) produces constraint

\[ \sum_i \left( \min(a_i \mod d, A \mod d) + \left\lfloor \frac{a_i}{d} \right\rfloor (A \mod d) \right) \ell_i \geq \left\lfloor \frac{A}{d} \right\rfloor (A \mod d) \]
Example Cut 2: Mixed Integer Rounding (MIR) Cut

Mixed integer rounding (MIR) cut [MW01] applied to (normalized) pseudo-Boolean constraint

\[ \sum_i a_i \ell_i \geq A \]

with divisor \( d \in \mathbb{N}^+ \) produces constraint

\[ \sum_i \left( \min(a_i \mod d, A \mod d) + \left\lfloor \frac{a_i}{d} \right\rfloor (A \mod d) \right) \ell_i \geq \left\lfloor \frac{A}{d} \right\rfloor (A \mod d) \]

Concretely, MIR cut with divisor 3 applied on

\[ x + 2y + 3z + 4w + 5u \geq 5 \]

yields

\[ x + 2y + 2z + 3w + 4u \geq 4 \]
Example Cut 2: Mixed Integer Rounding (MIR) Cut

Mixed integer rounding (MIR) cut [MW01] applied to (normalized) pseudo-Boolean constraint

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

with divisor $d \in \mathbb{N}^+$ produces constraint

$$\sum_i \left( \min(a_i \mod d, A \mod d) + \left\lfloor \frac{a_i}{d} \right\rfloor (A \mod d) \right) \ell_i \geq \left\lceil \frac{A}{d} \right\rceil (A \mod d)$$

Concretely, MIR cut with divisor 3 applied on

$$x + 2y + 3z + 4w + 5u \geq 5$$

yields

$$x + 2y + 2z + 3w + 4u \geq 4$$

For comparison, standard division by 3 and multiplication by 2 produces

$$2x + 2y + 2z + 4w + 4u \geq 4$$
Presolving

Topic for a separate talk (well, like everything else in this part...) Important for performance (but not as important as in CDCL?)
Presolving

Topic for a separate talk (well, like everything else in this part...) Important for performance (but not as important as in CDCL?)

Some simple (but efficient) techniques:

- **Substitution** of fixed variables
- **Normalization** of constraints: divide integer constraints by \( \text{gcd} \) on left-hand side and round on right-hand side
- **Probing**: tentatively assign binary variables and propagate
- **Dominance test**: remove constraints implied by other constraints
Presolving

Topic for a separate talk (well, like everything else in this part...) Important for performance (but not as important as in CDCL?)

Some simple (but efficient) techniques:

- **Substitution** of fixed variables
- **Normalization** of constraints: divide integer constraints by \( \text{gcd} \) on left-hand side and round on right-hand side
- **Probing**: tentatively assign binary variables and propagate
- **Dominance test**: remove constraints implied by other constraints

For more details, see talk by Gleixner [https://tinyurl.com/MIPtutorial](https://tinyurl.com/MIPtutorial)
MIP Conflict Analysis

MIP conflict analysis [Ach07] analogous to CDCL, but
- operate on clausal reasons extracted from constraints
- not on constraints themselves

Exponential loss in power!
MIP Conflict Analysis

MIP conflict analysis [Ach07] analogous to CDCL, but
- operate on clausal reasons extracted from constraints
- not on constraints themselves

Exponential loss in power!

Pigeonhole principle

\[
\begin{align*}
\sum_{j=1}^{n} x_{i,j} & \geq 1 & i & \in [n+1] \\
\sum_{i=1}^{n+1} x_{i,j} & \leq 1 & j & \in [n]
\end{align*}
\]

Conflict analysis with clausal reasons \(\Rightarrow\) indistinguishable from resolution on CNF encoding \(\Rightarrow\) exponential lower bound in [Hak85] applies
MIP Conflict Analysis

MIP conflict analysis [Ach07] analogous to CDCL, but
- operate on clausal reasons extracted from constraints
- **not** on constraints themselves

Exponential loss in power!

Pigeonhole principle

\[
\sum_{j=1}^{n} x_{i,j} \geq 1 \\
\sum_{i=1}^{n+1} x_{i,j} \leq 1
\]

Conflict analysis with clausal reasons \(\Rightarrow\) indistinguishable from resolution
on CNF encoding \(\Rightarrow\) exponential lower bound in [Hak85] applies

A bit stupid example... solved immediately, since LP relaxation infeasible
MIP Conflict Analysis

MIP conflict analysis [Ach07] analogous to CDCL, but

- operate on clausal reasons extracted from constraints
- **not** on constraints themselves

Exponential loss in power!

Pigeonhole principle

\[
\sum_{j=1}^{n} x_{i,j} \geq 1 \quad i \in [n+1]
\]
\[
\sum_{i=1}^{n+1} x_{i,j} \leq 1 \quad j \in [n]
\]

Conflict analysis with clausal reasons ⇒ indistinguishable from resolution on CNF encoding ⇒ exponential lower bound in [Hak85] applies

A bit stupid example... solved immediately, since LP relaxation infeasible

But can find other, more interesting benchmarks where MIP conflict analysis seems to suffer from this problem [DGN21]
Branching Heuristics

Dual gain

Given LP solution $x^*$, branch on $x_j$ such that $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
both provide good lower bound increase
Branching Heuristics

Dual gain

Given LP solution $x^*$, branch on $x_j$ such that $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
both provide good lower bound increase

Look ahead (strong branching)

- Consider all free variables $x_j$
- Solve LP for all branching decisions $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
- Pick best variable
Branching Heuristics

**Dual gain**

Given LP solution $x^*$, branch on $x_j$ such that $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$

both provide good lower bound increase

**Look ahead (strong branching)**

- Consider all free variables $x_j$
- Solve LP for all branching decisions $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
- Pick best variable

**Look back**

Compute estimate on gains based on past branching history (pseudo-costs)
Branching Heuristics

Dual gain

Given LP solution $x^*$, branch on $x_j$ such that $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
both provide good lower bound increase

Look ahead (strong branching)

- Consider all free variables $x_j$
- Solve LP for all branching decisions $x_j \geq \lceil x_j^* \rceil$ and $x_j \leq \lfloor x_j^* \rfloor$
- Pick best variable

Look back

Compute estimate on gains based on past branching history (pseudo-costs)

Keep also other statistics about variables to guide search
Node Selection

How to grow search tree?

- **Depth-first search (DFS)**: keeps cost for simplex calls small
- **Best bound search (BBS)**: Focus on improving lower bound (dual bound)
- **Best estimate search (BES)**: Focus on improving solution (primal bound)
Node Selection

How to grow search tree?

- **Depth-first search (DFS):** keeps cost for simplex calls small
- **Best bound search (BBS):** Focus on improving lower bound (dual bound)
- **Best estimate search (BES):** Focus on improving solution (primal bound)

Combine BBS and BES with **DFS plunges** to exploit simplex hot restarts
Primal Heuristics

- Improve solution (primal bound)
- Guide remaining search
Primal Heuristics

- Improve solution (primal bound)
- Guide remaining search

Example: Relaxation-enforced neighbourhood search

1. Solve LP relaxation to get $x^*$
2. Fix values of all $x_j$ such that $x_j^* \in \mathbb{N}$
3. For $x_j$ with fractional solution, reduce domain to $x_j \in \{\lfloor x_j^* \rfloor, \lceil x_j^* \rceil\}$
4. Solve new subproblem
Primal Heuristics

- Improve solution (primal bound)
- Guide remaining search

**Example: Relaxation-enforced neighbourhood search**

1. Solve LP relaxation to get $x^*$
2. Fix values of all $x_j$ such that $x_j^* \in \mathbb{N}$
3. For $x_j$ with fractional solution, reduce domain to $x_j \in \{\lfloor x_j^* \rfloor, \lceil x_j^* \rceil \}$
4. Solve new subproblem

Example of “fix-and-MIP” local neighbourhood search heuristic
(Interestingly, this turns ILP into 0-1 ILP subproblem)
And More...

1. Decomposition
   - Branch-and-price / column generation
   - Bender’s decomposition

2. Symmetry handling
   - Via graph automorphism
   - Or dedicated symmetry detection (commercial solvers)

3. Extended formulations (with new variables and constraints)

4. Parallelization

5. Restarts
Numerics and Correctness

Numerics

- Use floating point for efficiency reasons
- Can lead to rounding errors
- Exact MIP solvers like [CKSW13]
  - are significantly slower
  - don’t support the full range of state-of-the-art techniques
Numerics and Correctness

Numerics

- Use floating point for efficiency reasons
- Can lead to rounding errors
- Exact MIP solvers like [CKSW13]
  - are significantly slower
  - don’t support the full range of state-of-the-art techniques

Proof logging / certification

- Currently not available for state-of-the-art solvers
- Though known that even best commercial solvers sometimes give wrong results
- Some work on proof logging in [CGS17] — challenges:
  - How to capture wide diversity of techniques?
  - What is a convenient format?
  - How to generate proofs efficiently on-the-fly?
Some Interesting MIP Questions

1. Develop better heuristics to branch on general linear constraints (cf. stabbing planes [BFI+18])

2. Design stronger conflict analysis operating directly on linear constraints (borrow ideas from native pseudo-Boolean solvers?)

3. Provide rigorous understanding of MIP solver performance

4. Develop families of theory benchmarks and computational complexity results for them (cf. SAT solving and proof complexity [BN21])

5. Steal best MIP ideas and use for pseudo-Boolean solving?!
Some Interesting MIP Questions

1. Develop better heuristics to branch on general linear constraints (cf. stabbing planes [BFI+18])

2. Design stronger conflict analysis operating directly on linear constraints (borrow ideas from native pseudo-Boolean solvers?)

3. Provide rigorous understanding of MIP solver performance

4. Develop families of theory benchmarks and computational complexity results for them (cf. SAT solving and proof complexity [BN21])

5. Steal best MIP ideas and use for pseudo-Boolean solving?! [next and final topic]
Combining PB Solving and Mixed Integer Programming

Pseudo-Boolean solvers

- Sophisticated conflict analysis using cutting planes method
- Sometimes terrible performance even when LP relaxation infeasible [EGNV18]
Combining PB and MIP Techniques

Combining PB Solving and Mixed Integer Programming

**Pseudo-Boolean solvers**
- Sophisticated conflict analysis using cutting planes method
- Sometimes terrible performance even when LP relaxation infeasible [EGNV18]

**Mixed integer linear programming solvers**
- Powerful search
- Exploits information from LP relaxations
- Rich variety of cut generation routines
- But conflict analysis not so great...
Combining PB and MIP Techniques

Combining PB Solving and Mixed Integer Programming

Pseudo-Boolean solvers
- Sophisticated conflict analysis using cutting planes method
- Sometimes terrible performance even when LP relaxation infeasible [EGNV18]

Mixed integer linear programming solvers
- Powerful search
- Exploits information from LP relaxations
- Rich variety of cut generation routines
- But conflict analysis not so great . . .

Why not merge the two to get the best of both worlds of SAT-style conflict-driven search and MIP-style branch-and-cut?
Balance Time Allocation for PB and LP Solving?

High-level idea: Give pseudo-Boolean solver access to LP solver
Balance Time Allocation for PB and LP Solving?

High-level idea: Give pseudo-Boolean solver access to LP solver

First challenge:

1. Using LP solver as preprocessor not sufficient
   - PB formulas can have feasible LP relaxations
   - but quickly turn infeasible after just a couple of decisions
   - Some such benchmarks very hard for PB solvers [EGNV18]
Balance Time Allocation for PB and LP Solving?

High-level idea: Give pseudo-Boolean solver access to LP solver

First challenge:

1. Using LP solver as preprocessor not sufficient
   - PB formulas can have feasible LP relaxations
   - but quickly turn infeasible after just a couple of decisions
   - Some such benchmarks very hard for PB solvers [EGNV18]

2. Consulting LP solver before each variable decision impractical
   - PB solving based on rapid alternation of decisions and propagations
   - Solving an LP relaxation is orders of magnitude slower
Balance Time Allocation for PB and LP Solving?

High-level idea: Give pseudo-Boolean solver access to LP solver

First challenge:

1. Using LP solver as preprocessor not sufficient
   - PB formulas can have feasible LP relaxations
   - but quickly turn infeasible after just a couple of decisions
   - Some such benchmarks very hard for PB solvers [EGNV18]

2. Consulting LP solver before each variable decision impractical
   - PB solving based on rapid alternation of decisions and propagations
   - Solving an LP relaxation is orders of magnitude slower

Need to carefully balance time allocation for PB solver and LP solver
Backtracking from LP Infeasibility?

What to do if LP call shows LP relaxation infeasible under current trail?

- Obviously, PB solver should backtrack
- But can only do conflict analysis on violated PB constraint
- And PB solver blissfully unaware of any conflict...
Backtracking from LP Infeasibility?

What to do if LP call shows LP relaxation infeasible under current trail?

- Obviously, PB solver should backtrack
- But can only do conflict analysis on violated PB constraint
- And PB solver blissfully unaware of any conflict...

More subtle issue:

- Efficient LP solvers use inexact floating-point arithmetic
- How to incorporate into Boolean solver that must maintain perfectly sound reasoning?
Sharing of Cut Constraints?

Cut constraints from LP solver
- When LP relaxation feasible, MIP solver generates cut constraint to remove the found LP solution
- Should such constraints be shared with the PB solver?
Sharing of Cut Constraints?

Cut constraints from LP solver
- When LP relaxation feasible, MIP solver generates cut constraint to remove the found LP solution
- Should such constraints be shared with the PB solver?

Cut constraints from PB solver
- PB solvers learns new constraints at high rate from conflict analysis
- These learned constraints can also be viewed as cuts
- Should such constraints be passed from PB solver to LP solver?
Report on Attempted PB-LP Integration [DGN21]

1. Interleave incremental LP solving within conflict-driven PB search
   - Limit LP solver time by enforcing total #LP pivots ≤ #PB conflicts
   - Only run LP solver when this condition holds
   - Abort if > $P$ pivots in single LP call; but if so also double limit $P$ to avoid wasted LP calls in future

2. When LP solver detects that LP relaxation infeasible
   - Use Farkas' lemma to find linear combination of constraints violated by trail
   - Use this Farkas constraint as starting point for conflict analysis
   - Computed using exact arithmetic, so no rounding errors
   - But might not be violated — if so, ignore and continue PB search

3. When LP solver finds solution to LP relaxation
   - Generate MIP-style Gomory cut
   - Share constraint to tighten search space on both PB side and LP side
   - Try to use LP solution to guide PB search (e.g., variable decisions)

4. Also explore letting PB solver pass learned constraints to LP solver
Report on Attempted PB-LP Integration [DGN21]

1. Interleave incremental LP solving within conflict-driven PB search
   - Limit LP solver time by enforcing total #LP pivots $\leq$ #PB conflicts
   - Only run LP solver when this condition holds
   - Abort if $> P$ pivots in single LP call; but if so also double limit $P$ to avoid wasted LP calls in future

2. When LP solver detects that LP relaxation infeasible
   - Farkas’ lemma $\Rightarrow$ linear combination of constraints violated by trail
   - Use this Farkas constraint as starting point for conflict analysis
   - Computed using exact arithmetic, so no rounding errors
   - But might not be violated — if so, ignore and continue PB search
Combining PB and MIP Techniques

A Proof-of-Concept Hybrid PB-LP Solver

Report on Attempted PB-LP Integration [DGN21]

1. Interleave incremental LP solving within conflict-driven PB search
   - Limit LP solver time by enforcing total $\#$LP pivots $\leq \#$PB conflicts
   - Only run LP solver when this condition holds
   - Abort if $> P$ pivots in single LP call; but if so also double limit $P$ to avoid wasted LP calls in future

2. When LP solver detects that LP relaxation infeasible
   - Farkas’ lemma $\Rightarrow$ linear combination of constraints violated by trail
   - Use this Farkas constraint as starting point for conflict analysis
   - Computed using exact arithmetic, so no rounding errors
   - But might not be violated — if so, ignore and continue PB search

3. When LP solver finds solution to LP relaxation
   - Generate MIP-style Gomory cut
   - Share constraint to tighten search space on both PB side and LP side
   - Try to use LP solution to guide PB search (e.g., variable decisions)
Interleave incremental LP solving within conflict-driven PB search

1. Limit LP solver time by enforcing \( \text{total } \#\text{LP pivots} \leq \#\text{PB conflicts} \)
2. Only run LP solver when this condition holds
3. Abort if \( > P \) pivots in single LP call; but if so also double limit \( P \) to avoid wasted LP calls in future

When LP solver detects that LP relaxation infeasible

4. Farkas’ lemma \( \Rightarrow \) linear combination of constraints violated by trail
5. Use this Farkas constraint as starting point for conflict analysis
6. Computed using exact arithmetic, so no rounding errors
7. But might not be violated — if so, ignore and continue PB search

When LP solver finds solution to LP relaxation

8. Generate MIP-style Gomory cut
9. Share constraint to tighten search space on both PB side and LP side
10. Try to use LP solution to guide PB search (e.g., variable decisions)

Also explore letting PB solver pass learned constraints to LP solver
Pseudo-Boolean Farkas Lemma

Given

- Pseudo-Boolean formula \( F = \{ C_1, \ldots, C_m \} \),
- partial assignment \( \rho \),

such that LP relaxation of residual formula \( F \upharpoonright \rho \) infeasible

Then \( \exists \) coefficients \( k_i \in \mathbb{N} \) such that linear combination

\[
\sum_{i=1}^{m} k_i \cdot C_i
\]

is violated by \( \rho \), i.e.,

\[
\text{slack}(\sum_{i=1}^{m} k_i \cdot C_i; \rho) < 0
\]

Observed in [MM04] that \( \sum_{i=1}^{m} k_i \cdot C_i \) is valid starting point for pseudo-Boolean conflict analysis
Relation to MIP Solvers with Conflict Analysis?

MIP solvers also combine constraint propagation and SAT-style clause learning with LP solving

- Implemented in SCIP [ABKW08]
- And also in closed-source solvers (see [AW13])

Important to understand similarities and differences — let’s give high-level description of PB search and conflict analysis phrased in MIP language
Relation to MIP Solvers with Conflict Analysis?

MIP solvers also combine constraint propagation and SAT-style clause learning with LP solving
- Implemented in SCIP [ABKW08]
- And also in closed-source solvers (see [AW13])

Important to understand similarities and differences — let’s give high-level description of PB search and conflict analysis phrased in MIP language

**Pseudo-Boolean search**

1. Make **decision** to assign free variable to 0 or 1
2. **Propagate** all assignments implied by some linear constraint until saturation
3. If no contradiction, go to step 1
4. Otherwise some constraint $C$ violated ⇒ trigger conflict analysis
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find **reason constraint** \( R \) responsible for propagating last variable \( x \) in \( C \) to “wrong value”
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find reason constraint $R$ responsible for propagating last variable $x$ in $C$ to “wrong value”

2. Apply division/saturation to generate cut $R_{cut}$ propagating $x$ to $\{0, 1\}$-value (over the reals)
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find *reason constraint* $R$ responsible for propagating last variable $x$ in $C$ to “wrong value”

2. Apply division/saturation to generate cut $R_{\text{cut}}$ propagating $x$ to $\{0, 1\}$-value (over the reals)

3. Set $D :=$ smallest integer linear combination of $R_{\text{cut}}$ and $C$ for which $x$ cancels — $D$ violated by current solvers assignment with $x$ removed
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find reason constraint $R$ responsible for propagating last variable $x$ in $C$ to “wrong value”

2. Apply division/saturation to generate cut $R_{\text{cut}}$ propagating $x$ to \{0, 1\}-value (over the reals)

3. Set $D :=$ smallest integer linear combination of $R_{\text{cut}}$ and $C$ for which $x$ cancels — $D$ violated by current solvers assignment with $x$ removed

4. Unless $D$ satisfies termination criterion (assertiveness), set $C := D$ and go to step 1
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find reason constraint $R$ responsible for propagating last variable $x$ in $C$ to “wrong value”
2. Apply division/saturation to generate cut $R_{\text{cut}}$ propagating $x$ to $\{0, 1\}$-value (over the reals)
3. Set $D :=$ smallest integer linear combination of $R_{\text{cut}}$ and $C$ for which $x$ cancels — $D$ violated by current solvers assignment with $x$ removed
4. Unless $D$ satisfies termination criterion (assertiveness), set $C := D$ and go to step 1
5. Learn assertive $D$, i.e., add to solver database of constraints
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find reason constraint \( R \) responsible for propagating last variable \( x \) in \( C \) to “wrong value”
2. Apply division/saturation to generate cut \( R_{\text{cut}} \) propagating \( x \) to \( \{0, 1\} \)-value (over the reals)
3. Set \( D := \) smallest integer linear combination of \( R_{\text{cut}} \) and \( C \) for which \( x \) cancels — \( D \) violated by current solvers assignment with \( x \) removed
4. Unless \( D \) satisfies termination criterion (assertiveness), set \( C := D \) and go to step 1
5. Learn assertive \( D \), i.e., add to solver database of constraints
6. Backjump by undoing further assignments in reverse chronological order until \( D \) is no longer violated
PB Conflict Analysis “in MIP Language”

Pseudo-Boolean conflict analysis (simplified description)

1. Find reason constraint $R$ responsible for propagating last variable $x$ in $C$ to “wrong value”
2. Apply division/saturation to generate cut $R_{cut}$ propagating $x$ to $\{0, 1\}$-value (over the reals)
3. Set $D :=$ smallest integer linear combination of $R_{cut}$ and $C$ for which $x$ cancels — $D$ violated by current solvers assignment with $x$ removed
4. Unless $D$ satisfies termination criterion (assertiveness), set $C := D$ and go to step 1
5. Learn assertive $D$, i.e., add to solver database of constraints
6. Backjump by undoing further assignments in reverse chronological order until $D$ is no longer violated
7. Switch back to search phase
Comparison to MIP Propagation and Conflict Analysis

Propagation in SCIP

- Fast, simple propagation in PB solvers
- Plus powerful, but slower, method of solving LP relaxations
Comparison to MIP Propagation and Conflict Analysis

Propagation in SCIP

- Fast, simple propagation in PB solvers
- Plus powerful, but slower, method of solving LP relaxations

Conflict analysis in SCIP [Ach07]

- Perform derivation not on reason constraints $R$ as described above
- Instead use disjunctive clauses extracted from reason constraints
- Incurs exponential loss in reasoning power compared to operating on actual linear constraints (follows from [BKS04, CCT87, Hak85])
Comparison to MIP Propagation and Conflict Analysis

Propagation in SCIP
- Fast, simple propagation in PB solvers
- Plus powerful, but slower, method of solving LP relaxations

Conflict analysis in SCIP [Ach07]
- Perform derivation not on reason constraints $R$ as described above
- Instead use disjunctive clauses extracted from reason constraints
- Incurs exponential loss in reasoning power compared to operating on actual linear constraints (follows from [BKS04, CCT87, Hak85])

Arithmetic
- SCIP uses floating point
- Reasoning steps in PB solver computed with exact integer arithmetic
- No issues with possible rounding errors
**RoundingSat** (RS) enhanced with
- LP solver
  *SoPlex* (SPX) 
  (from SCIP)
- Gomory cuts (GC)
- shared learned PB cuts (LC)

compared to other solvers

**Experimental Results for Knapsack Benchmarks** [Pis05]

- **RoundingSat** (RS)
- **SoPlex** (SPX)
- Gomory cuts (GC)
- shared learned PB cuts (LC)

**Knapsack**

(higher is better, 783 instances)

<table>
<thead>
<tr>
<th>Solver</th>
<th>Number of solved instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCIP</td>
<td>765</td>
</tr>
<tr>
<td>RS</td>
<td>520</td>
</tr>
<tr>
<td>RS+SPX</td>
<td>600</td>
</tr>
<tr>
<td>RS+SPX+GC</td>
<td>670</td>
</tr>
<tr>
<td>RS+SPX+GC+LC</td>
<td>680</td>
</tr>
<tr>
<td>Sat4J</td>
<td>374</td>
</tr>
<tr>
<td>NaPS</td>
<td>111</td>
</tr>
</tbody>
</table>

**Timeout limit (s)**

- 0
- 100
- 200
- 300
- 400
- 500
- 600
- 700

**Number of solved instances**

- Knapsack (higher is better, 783 instances)
- SCIP (765 solved)
- RS (520)
- RS+SPX (600)
- RS+SPX+GC (670)
- RS+SPX+GC+LC (680)
- Sat4J (374)
- NaPS (111)
Experimental Results for PB and MIPLIB Benchmarks

**RoundingSat (RS)** run on PB and 0-1 ILP instances with
- LP solver (+SPX)
- plus Gomory cuts (+GC)
- plus sharing cuts learned by PB solver (+LC)

compared to other solvers

# instances solved (to optimality for optimization problems)

Highlighting **1st, 2nd, and 3rd** best
Combining PB and MIP Techniques

Evaluation and Conclusions

Experimental Results for PB and MIPLIB Benchmarks

**RoundingSat (RS)** run on PB and 0-1 ILP instances with
- LP solver (**+SPX**)
- plus Gomory cuts (**+GC**)
- plus sharing cuts learned by PB solver (**+LC**)

compared to other solvers

# instances solved (to optimality for optimization problems)
Highlighting 1st, 2nd, and 3rd best

<table>
<thead>
<tr>
<th></th>
<th>SCIP</th>
<th>RS</th>
<th>+SPX</th>
<th>+GC</th>
<th>+LC</th>
<th>SAT4J</th>
<th>NAPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>PB16dec (1783)</td>
<td>1123</td>
<td>1472</td>
<td>1453</td>
<td>1452</td>
<td>1451</td>
<td>1432</td>
<td>1400</td>
</tr>
<tr>
<td>PB16opt (1600)</td>
<td>1057</td>
<td>862</td>
<td>988</td>
<td>986</td>
<td>993</td>
<td>776</td>
<td>896</td>
</tr>
<tr>
<td>MIPdec (556)</td>
<td>264</td>
<td>203</td>
<td>263</td>
<td>261</td>
<td>259</td>
<td>169</td>
<td>170</td>
</tr>
<tr>
<td>MIPopt (291)</td>
<td>125</td>
<td>78</td>
<td>101</td>
<td>102</td>
<td>102</td>
<td>62</td>
<td>65</td>
</tr>
</tbody>
</table>

Jakob Nordström (UCPH & LU) Pseudo-Boolean Solving and Optimization Simons Institute Feb '21 116/121
Performance of Integrated PB-LP Solver

Best of both worlds?

- At least well-rounded performance
- Hybrid PB-LP solver always competitive with best solver
- Pretty dramatic improvements for optimization problems compared to pseudo-Boolean state of the art
- ... But SCIP is hard to beat
Performance of Integrated PB-LP Solver

1. **Best of both worlds?**
   - At least well-rounded performance
   - Hybrid PB-LP solver always competitive with best solver
   - Pretty dramatic improvements for optimization problems compared to pseudo-Boolean state of the art
   - ... But SCIP is hard to beat

2. **Adding LP solving causes performance loss on PB decision instances**
   - Worse results on satisfiable instances
   - Better search (lower conflict count) but slower — doesn’t pay off in terms of running time
**Performance of Integrated PB-LP Solver**

1. **Best of both worlds?**
   - At least well-rounded performance
   - Hybrid PB-LP solver always competitive with best solver
   - Pretty dramatic improvements for optimization problems compared to pseudo-Boolean state of the art
   - … But SCIP is hard to beat

2. **Adding LP solving causes performance loss on PB decision instances**
   - Worse results on satisfiable instances
   - Better search (lower conflict count) but slower — doesn’t pay off in terms of running time

3. **Sharing Gomory cuts and learned cuts not so helpful**
   - Except for knapsack benchmarks, where they help a lot
   - And maybe we could/should fine-tune how sharing is done?
Usefulness/Usage of Constraints

Estimate usefulness of different types of constraints

- Proxy: how often used in conflict analysis?
- Certainly not perfect measure
- But hopefully tells us something interesting
Usefulness/Usage of Constraints

**Estimate usefulness of different types of constraints**
- Proxy: how often used in conflict analysis?
- Certainly not perfect measure
- But hopefully tells us something interesting

**Farkas constraints**
- More useful than regular learned constraints for optimization problems
- Not so for decision problems
Usefulness/Usage of Constraints

Estimate usefulness of different types of constraints

- Proxy: how often used in conflict analysis?
- Certainly not perfect measure
- But hopefully tells us something interesting

Farkas constraints

- More useful than regular learned constraints for optimization problems
- Not so for decision problems

Constraints learned after Farkas-based conflicts

- Less useful than regular learned constraints
- But big spread in usage measurements
Future Research Directions for PB-LP Integration (1/2)

1. **Fine-tune heuristics**
   - Improved LP-based cut generation?
   - Smarter sharing of PB constraints with LP solver?
   - Dynamic allocation of PB and LP solving time based on contributions?

2. Understand better how constraints from LP solver contribute.
   - Why are Farkas constraints so useful?
   - But constraints learned from Farkas constraint conflicts not useful?

3. Make more intelligent use in PB solver of information from solutions to LP relaxations.

4. Use MIP presolving in pseudo-Boolean solvers.

5. Use MIR cuts and/or other MIP cut rules to improve pseudo-Boolean conflict analysis.
Future Research Directions for PB-LP Integration (1/2)

1. Fine-tune heuristics
   - Improved LP-based cut generation?
   - Smarter sharing of PB constraints with LP solver?
   - Dynamic allocation of PB and LP solving time based on contributions?

2. Understand better how constraints from LP solver contribute
   - Why are Farkas constraints so useful?
   - But constraints learned from Farkas constraint conflicts not useful?
Future Research Directions for PB-LP Integration (1/2)

1. Fine-tune heuristics
   - Improved LP-based cut generation?
   - Smarter sharing of PB constraints with LP solver?
   - Dynamic allocation of PB and LP solving time based on contributions?

2. Understand better how constraints from LP solver contribute
   - Why are Farkas constraints so useful?
   - But constraints learned from Farkas constraint conflicts not useful?

3. Make more intelligent use in PB solver of information from solutions to LP relaxations
Future Research Directions for PB-LP Integration (1/2)

1. Fine-tune heuristics
   - Improved LP-based cut generation?
   - Smarter sharing of PB constraints with LP solver?
   - Dynamic allocation of PB and LP solving time based on contributions?

2. Understand better how constraints from LP solver contribute
   - Why are Farkas constraints so useful?
   - But constraints learned from Farkas constraint conflicts not useful?

3. Make more intelligent use in PB solver of information from solutions to LP relaxations

4. Use MIP presolving in pseudo-Boolean solvers
Future Research Directions for PB-LP Integration (1/2)

1. Fine-tune heuristics
   - Improved LP-based cut generation?
   - Smarter sharing of PB constraints with LP solver?
   - Dynamic allocation of PB and LP solving time based on contributions?

2. Understand better how constraints from LP solver contribute
   - Why are Farkas constraints so useful?
   - But constraints learned from Farkas constraint conflicts not useful?

3. Make more intelligent use in PB solver of information from solutions to LP relaxations

4. Use MIP presolving in pseudo-Boolean solvers

5. Use MIR cuts and/or other MIP cut rules to improve pseudo-Boolean conflict analysis
Combine LP solver with core-guided search or IHS approach
Future Research Directions for PB-LP Integration (1/2)

6. Combine LP solver with core-guided search or IHS approach

7. Improve pseudo-Boolean search
   - \textit{RoundingSat} with LP integration or core-guided search seems to be state of the art for PB solving
   - But solver much better on unsatisfiable instances (proving optimality) than on satisfiable ones (finding solutions)
Future Research Directions for PB-LP Integration (1/2)

6. Combine LP solver with core-guided search or IHS approach

7. Improve pseudo-Boolean search
   - RUNDINGSat with LP integration or core-guided search seems to be state of the art for PB solving
   - But solver much better on unsatisfiable instances (proving optimality) than on satisfiable ones (finding solutions)

8. Export pseudo-Boolean conflict analysis to MIP
Future Research Directions for PB-LP Integration (1/2)

6. Combine LP solver with core-guided search or IHS approach

7. Improve pseudo-Boolean search
   - RoundingSat with LP integration or core-guided search seems to be state of the art for PB solving
   - But solver much better on unsatisfiable instances (proving optimality) than on satisfiable ones (finding solutions)

8. Export pseudo-Boolean conflict analysis to MIP

9. Use hybrid PB-LP solver to solve 0-1 MIP problems
   - PB solver decides on Boolean variables and propagates
   - LP solver takes care of real-valued variables
Summing up

- **Pseudo-Boolean optimization** powerful and expressive framework
- Can be attacked with methods from
  - SAT solving and MaxSAT solving
  - “Native” cutting-planes-based pseudo-Boolean reasoning
  - Mixed integer linear programming
- Approaches with complementary strengths — room for synergies?
- Some highly nontrivial challenges regarding
  - Algorithm design
  - Efficient implementation
  - Theoretical understanding
- But maybe also quite a bit of low-hanging fruit?
- And in any case *lots of fun questions to work on!* 😊
Summing up

- **Pseudo-Boolean optimization** powerful and expressive framework
- Can be attacked with methods from
  - SAT solving and MaxSAT solving
  - “Native” cutting-planes-based pseudo-Boolean reasoning
  - Mixed integer linear programming
- Approaches with complementary strengths — room for synergies?
- Some highly nontrivial challenges regarding
  - Algorithm design
  - Efficient implementation
  - Theoretical understanding
- But maybe also quite a bit of low-hanging fruit?
- And in any case *lots of fun questions to work on!* 😊

Thank you for your attention!
References I


References III


References V


References VII


References IX


References XI


References XIV


References XV


References XVI


References XVII


