Computing with SAT Oracles

Joao Marques-Silva

University of Lisbon, Portugal

VMCAI 2019 Winter School
IST, Lisbon, Portugal
January 09-12 2019
What is SAT?

- **SAT** is the decision problem for **propositional logic**
  - Well-formed **propositional formulas**, with variables, logical connectives: ¬, ∧, ∨, →, ↔, and parenthesis: (, )
  - Often restricted to **Conjunctive Normal Form (CNF)**
What is SAT?

- **SAT** is the decision problem for propositional logic
  - Well-formed propositional formulas, with variables, logical connectives: $\neg, \land, \lor, \to, \leftrightarrow$, and parenthesis: $(, )$
  - Often restricted to **Conjunctive Normal Form (CNF)**
  - **Goal:** Decide whether formula has a satisfying assignment
What is SAT?

- **SAT** is the **decision problem for propositional logic**
  - Well-formed **propositional formulas**, with variables, logical connectives: ¬, ∧, ∨, →, ↔, and parenthesis: (, )
  - Often restricted to **Conjunctive Normal Form (CNF)**
  - **Goal:**
    
    Decide whether formula has a satisfying assignment

- **SAT** is **NP-complete**

[Coo71]
The CDCL SAT disruption

- CDCL SAT solving is a success story of Computer Science
• **CDCL SAT solving** is a **success story** of Computer Science
  - Conflict-Driven Clause Learning (**CDCL**)
  - (CDCL) SAT has impacted many different fields
  - Hundreds (thousands?) of practical applications
CDCL SAT solver improvement

[Source: Simon 2015]
How good are SAT solvers?

Demos

1. POSIT: state of the art DPLL SAT solver in 1995
2. GRASP: first CDCL SAT solver, state of the art 1995
3. Minisat: CDCL SAT solver, state of the art until the late 00s
4. Glucose: modern state of the art CDCL SAT solver

Example 1: model checking example (from IBM)
Example 2: cooperative path finding (CPF)
How good are SAT solvers?

Demos

• Sample SAT of solvers:
  1. **POSIT**: state of the art DPLL SAT solver in 1995
  2. **GRASP**: first CDCL SAT solver, state of the art 1995~2000
  3. **Minisat**: CDCL SAT solver, state of the art until the late 00s
  4. **Glucose**: modern state of the art CDCL SAT solver
  5. ...
How good are SAT solvers?

Demos

- Sample SAT of solvers:
  1. **POSIT**: state of the art DPLL SAT solver in 1995
  2. **GRASP**: first CDCL SAT solver, state of the art 1995–2000
  3. **Minisat**: CDCL SAT solver, state of the art until the late 00s
  4. **Glucose**: modern state of the art CDCL SAT solver
  5. ...

- **Example 1**: model checking example (from IBM)
- **Example 2**: cooperative path finding (CPF)
How good are SAT solvers?

- Number of seconds since the Big Bang: \( \approx 10^{17} \)
How good are SAT solvers?

- Number of seconds since the Big Bang: $\approx 10^{17}$

- Number of fundamental particles in observable universe: $\approx 10^{80}$
  (or $\approx 10^{85}$)
How good are SAT solvers?

- Number of seconds since the Big Bang: \( \approx 10^{17} \)

- Number of fundamental particles in observable universe: \( \approx 10^{80} \)
  (or \( \approx 10^{85} \))

- Search space with 15775 propositional variables (worst case):
How good are SAT solvers?

- Number of seconds since the Big Bang: \( \approx 10^{17} \)

- Number of fundamental particles in observable universe: \( \approx 10^{80} \) (or \( \approx 10^{85} \))

- Search space with 15775 propositional variables (worst case):
  - \# of assignments to 15775 variables: \( > 10^{4748} \)!
  - **Obs:** SAT solvers in the late 90s (but formula dependent)
How good are SAT solvers?

- Number of seconds since the Big Bang: \( \approx 10^{17} \)

- Number of fundamental particles in observable universe: \( \approx 10^{80} \)
  (or \( \approx 10^{85} \))

- Search space with 15775 propositional variables (worst case):
  - \# of assignments to 15775 variables: \( > 10^{4748} \)!
  - **Obs:** SAT solvers in the late 90s (but formula dependent)

- Search space with 2832875 propositional variables (worst case):
How good are SAT solvers?

- Number of seconds since the Big Bang: $\approx 10^{17}$

- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)

- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ !
  - Obs: SAT solvers in the late 90s (but formula dependent)

- Search space with 2832875 propositional variables (worst case):
  - # of assignments to $> 2.8 \times 10^6$ variables: $\gg 10^{840000}$ !!
  - Obs: SAT solvers at present (but formula dependent)
SAT can make the difference – axiom pinpointing

- $\mathcal{EL}^+$ medical ontologies
  - Minimal unsatisfiability (MUSes) & maximal satisfiability (MCSes)
    & Enumeration

[AMM15]
• Model-based diagnosis problem instances
  – Maximum satisfiability (MaxSAT)
CDCL SAT is ubiquitous in problem solving.
This tutorial

- Part #0: Basic definitions & notation
This tutorial

- Part #0: Basic definitions & notation
- Part #1: Modern SAT solvers
  - Conflict-Driven Clause Learning (CDCL) SAT solvers
    - Goal: Overview for non-experts
This tutorial

• Part #0: Basic definitions & notation

• Part #1: Modern SAT solvers
  – **Conflict-Driven Clause Learning (CDCL)** SAT solvers
  
     ▶ **Goal:** Overview for non-experts

• Part #2: Modeling problems for SAT
  – Propositional encodings
  – Modeling examples

• Part #3: Problem solving with SAT oracles
  – Minimal unsatisfiability (MUS)
  – Maximum satisfiability (MaxSAT)
  – Maximal satisfiability (MSS/MCS); Enumeration problems
  – Quantification problems; Counting problems; Etc.

• Part #4: Sample of applications

• Part #5: A glimpse of the future
This tutorial

• Part #0: Basic definitions & notation
• Part #1: Modern SAT solvers
  – Conflict-Driven Clause Learning (CDCL) SAT solvers
    ▶ Goal: Overview for non-experts
• Part #2: Modeling problems for SAT
  – Propositional encodings
  – Modeling examples
• Part #3: Problem solving with SAT oracles
  – Minimal unsatisfiability (MUS)
  – Maximum satisfiability (MaxSAT)
  – Maximal satisfiability (MSS/MCS); Enumeration problems
  – Quantification problems; Counting problems; Etc.
This tutorial

• Part #0: Basic definitions & notation

• Part #1: Modern SAT solvers
  – **Conflict-Driven Clause Learning (CDCL)** SAT solvers
    ▶ **Goal:** Overview for non-experts

• Part #2: Modeling problems for SAT
  – Propositional encodings
  – Modeling examples

• Part #3: Problem solving with SAT oracles
  – Minimal unsatisfiability (**MUS**)
  – Maximum satisfiability (**MaxSAT**)
  – Maximal satisfiability (MSS/MCS); Enumeration problems
  – Quantification problems; Counting problems; Etc.

• Part #4: Sample of applications
This tutorial

• Part #0: Basic definitions & notation

• Part #1: Modern SAT solvers
  – Conflict-Driven Clause Learning (CDCL) SAT solvers
    ▶ **Goal:** Overview for non-experts

• Part #2: Modeling problems for SAT
  – Propositional encodings
  – Modeling examples

• Part #3: Problem solving with SAT oracles
  – Minimal unsatisfiability (MUS)
  – Maximum satisfiability (MaxSAT)
  – Maximal satisfiability (MSS/MCS); Enumeration problems
  – Quantification problems; Counting problems; Etc.

• Part #4: Sample of applications

• Part #5: A glimpse of the future
Part 0

Basic Definitions
Preliminaries

- **Variables**: \( w, x, y, z, a, b, c, \ldots \)
- **Literals**: \( w, \bar{x}, \bar{y}, a, \ldots \), but also \( \neg w, \neg y, \ldots \)
- **Clauses**: disjunction of literals or set of literals
- **Formula**: conjunction of clauses or set of clauses
- **Model (satisfying assignment)**: partial/total mapping from variables to \( \{0, 1\} \) that satisfies formula
- **Each clause can be satisfied**, **falsified**, but also **unit**, **unresolved**
- **Formula can be** **SAT/UNSAT**
Preliminaries

- Variables: \( w, x, y, z, a, b, c, \ldots \)
- Literals: \( w, \overline{x}, \overline{y}, a, \ldots \), but also \( \neg w, \neg y, \ldots \)
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to \( \{0, 1\} \) that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT
- Example:

\[
\mathcal{F} \triangleq (r) \land (\overline{r} \lor s) \land (\overline{w} \lor a) \land (\overline{x} \lor b) \land (\overline{y} \lor \overline{z} \lor c) \land (\overline{b} \lor \overline{c} \lor d)
\]

- Example models:
Preliminaries

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT
- Example:

$$
\mathcal{F} \triangleq (r) \land (\bar{r} \lor s) \land (\bar{w} \lor a) \land (\bar{x} \lor b) \land (\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d)
$$

- Example models:
  - $\{r, s, a, b, c, d\}$
Preliminaries

• Variables: \( w, x, y, z, a, b, c, \ldots \)
• Literals: \( w, \overline{x}, \overline{y}, a, \ldots \), but also \( \neg w, \neg y, \ldots \)
• Clauses: disjunction of literals or set of literals
• Formula: conjunction of clauses or set of clauses
• Model (satisfying assignment): partial/total mapping from variables to \( \{0, 1\} \) that satisfies formula
• Each clause can be satisfied, falsified, but also unit, unresolved
• Formula can be SAT/UNSAT
• Example:

\[
\mathcal{F} \triangleq (r) \land (\overline{r} \lor s) \land (\overline{w} \lor a) \land (\overline{x} \lor b) \land (\overline{y} \lor \overline{z} \lor c) \land (\overline{b} \lor \overline{c} \lor d)
\]

- Example models:
  - \( \{r, s, a, b, c, d\} \)
  - \( \{r, s, \overline{x}, y, \overline{w}, z, \overline{a}, b, c, d\} \)
Resolution

- Resolution rule:

\[
\frac{(\alpha \lor x) \quad (\beta \lor \overline{x})}{(\alpha \lor \beta)}
\]

- Complete proof system for propositional logic

[DP60, Rob65]
Resolution

- Resolution rule:

\[
\frac{(\alpha \lor x) \quad (\beta \lor \bar{x})}{\alpha \lor \beta}
\]

- Complete proof system for propositional logic

- Extensively used with (CDCL) SAT solvers
Resolution

- **Resolution rule:**

\[
\frac{(\alpha \lor x) \quad (\beta \lor \overline{x})}{(\alpha \lor \beta)}
\]

- Complete proof system for propositional logic

\[
\begin{align*}
(x \lor a) & \quad (\overline{x} \lor a) \\
\hline
(a) & \quad (\overline{a})
\end{align*}
\]

\[
\overline{\bot}
\]

- Extensively used with (CDCL) SAT solvers

- **Self-subsuming resolution** (with \(\alpha' \subseteq \alpha\)):

\[
\frac{(\alpha \lor x) \quad (\alpha' \lor \overline{x})}{(\alpha)}
\]

- \((\alpha)\) subsumes \((\alpha \lor x)\)

[DP60, Rob65]

[SP04, SB09]
Unit propagation

\[ \mathcal{F} = (r) \land (\bar{r} \lor s) \land \\
(\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \land \\
(\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]
Unit propagation

\[ F = (r) \land (\bar{r} \lor s) \land (\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \land (\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]

- Decisions / Variable Branchings:
  \[ w = 1, x = 1, y = 1, z = 1 \]
Unit propagation

\[ \mathcal{F} = (r) \land (\bar{r} \lor s) \land (\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \land (\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]

- Decisions / Variable Branchings:
  \( w = 1, \ x = 1, \ y = 1, \ z = 1 \)

- **Unit clause rule**: if clause is unit, its sole literal **must** be satisfied
Unit propagation

\[ \mathcal{F} = (r) \land (\bar{r} \lor s) \land (\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \land (\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]

- Decisions / Variable Branchings:
  \[ w = 1, x = 1, y = 1, z = 1 \]

- **Unit clause rule:** if clause is unit, its sole literal must be satisfied
Unit propagation

\[ \mathcal{F} = (r) \land (\bar{r} \lor s) \land (\bar{w} \lor a) \land (\bar{x} \lor \bar{a} \lor b) \land (\bar{y} \lor \bar{z} \lor c) \land (\bar{b} \lor \bar{c} \lor d) \]

- **Decisions / Variable Branchings:**
  \[ w = 1, \; x = 1, \; y = 1, \; z = 1 \]

- **Unit clause rule:** if clause is unit, its sole literal **must** be satisfied

- **Additional definitions:**
  - **Antecedent (or reason)** of an implied assignment
    \[ (\bar{b} \lor \bar{c} \lor d) \text{ for } d \]
  - **Associate assignment with decision levels**
    \[ w = 1 \circ 1, \; x = 1 \circ 2, \; y = 1 \circ 3, \; z = 1 \circ 4 \]
    \[ r = 1 \circ 0, \; d = 1 \circ 4, \; ... \]
Resolution proofs

- Refutation of unsatisfiable formula by iterated resolution operations produces resolution proof
- An example:
  \[ \mathcal{F} = (\overline{c}) \land (\overline{b}) \land (\overline{a} \lor c) \land (a \lor b) \land (a \lor \overline{d}) \land (\overline{a} \lor \overline{d}) \]
- Resolution proof:

```
\frac{(a \lor b)}{(\overline{b})} \frac{(\overline{a} \lor c)}{(b \lor c)} \frac{(\overline{c})}{\bot}
```

- A modern SAT solver can generate resolution proofs using clauses learned by the solver

[ZM03]
Unsatisfiable cores & proof traces

- CNF formula:

\[ \mathcal{F} = (\overline{c}) \land (\overline{b}) \land (\overline{a} \lor c) \land (a \lor b) \land (a \lor \overline{d}) \land (\overline{a} \lor \overline{d}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
</table>
| 0     | \emptyset | \overline{b} \rightarrow a | \\
|       |      | \overline{c} \rightarrow \bot |

Implication graph with conflict
Unsatisfiable cores & proof traces

- CNF formula:

\[ \mathcal{F} = (\bar{c}) \land (\bar{b}) \land (\bar{a} \lor c) \land (a \lor b) \land (a \lor \bar{d}) \land (\bar{a} \lor \bar{d}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>(\bar{b} \rightarrow a)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\bar{c} \rightarrow \bot)</td>
</tr>
</tbody>
</table>

Proof trace \(\bot\): \((\bar{a} \lor c) \ (a \lor b) \ (\bar{c}) \ (\bar{b})\)
Unsatisfiable cores & proof traces

- CNF formula:

$$\mathcal{F} = (\overline{c}) \land (\overline{b}) \land (\overline{a} \lor c) \land (a \lor b) \land (a \lor \overline{d}) \land (\overline{a} \lor \overline{d})$$

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td>$\overline{b} \rightarrow a$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\overline{c} \rightarrow \bot$</td>
</tr>
</tbody>
</table>

Resolution proof follows structure of conflicts
Unsatisfiable cores & proof traces

• CNF formula:

\[ \mathcal{F} = (\overline{c}) \land (\overline{b}) \land (\overline{a} \lor c) \land (a \lor b) \land (a \lor \overline{d}) \land (\overline{a} \lor \overline{d}) \]


Unsatisfiable subformula (core): \((\overline{c}), (\overline{b}), (\overline{a} \lor c), (a \lor b)\)
The DPLL algorithm

- Optional: pure literal rule
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

- Optional: pure literal rule
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

- Optional: pure literal rule
The DPLL algorithm

• Optional: pure literal rule

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( x )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( y )</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>( \bar{a} )</td>
<td>( \bar{b} )</td>
</tr>
</tbody>
</table>
The DPLL algorithm

Unassigned variables?
Y
Assign value to variable
N
Satisfiable

Y

Conflict?
N
Can undo decision?
N
Unsatisfiable
Y
Backtrack & flip variable

Optional: pure literal rule

F = (x ∨ y) ∧ (a ∨ b) ∧ (ā ∨ b) ∧ (a ∨ ā) ∧ (ā ∨ ā)

Level Dec. Unit Prop.
0 ∅
1 x
2 ā
3 a → b → ⊥
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor b) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\emptyset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(x)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(\bar{y})</td>
<td></td>
</tr>
</tbody>
</table>
| 3     | \(\bar{a}\) |\(\bar{b}\)| \(\bot\)

- **Optional:** pure literal rule
The DPLL algorithm

\[ F = (x \lor y) \land (a \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

- **Level Dec. Unit Prop.**
  - 0: \( \emptyset \)
  - 1: \( \bar{x} \rightarrow y \)
  - 2: \( a \rightarrow b \rightarrow \bot \)

- **Optional:** pure literal rule

- Unassigned variables?
  - Y: Assign value to variable
  - N: Unit propagation

- Conflict?
  - Y: Can undo decision?
  - N: Unassignable

- Satisfiable

- Backtrack & flip variable

- F = \((x \lor y) \land (a \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b})\)
The DPLL algorithm

\[ \mathcal{F} = (x \lor y) \land (a \lor b) \land (\bar{a} \lor \bar{b}) \land (a \lor \bar{b}) \land (\bar{a} \lor \bar{b}) \]

- Level 0: \( \emptyset \)
- Level 1: \( \bar{x} \rightarrow y \)
- Level 2: \( \bar{a} \rightarrow \bar{b} \rightarrow \bot \)

- Optional: pure literal rule
Part 1

CDCL SAT Solving
What is a CDCL SAT solver?

- Extend DPLL SAT solver with: [DP60, DLL62]
  - Clause learning & non-chronological backtracking [MS95, MSS96b, MSS99]
  - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
  - Lazy data structures
  - Conflict-guided branching

- ...
What is a CDCL SAT solver?

- Extend **DPLL SAT solver** with: [DP60, DLL62]
  - Clause learning & non-chronological backtracking [MS95, MSS96b, MSS99]
    - Exploit UIPs [MS95, MSS99, ZMMM01, SSS12]
    - Minimize learned clauses [SB09, Gel09, LLX+17]
    - Opportunistically delete clauses [MSS96b, MSS99, GN02, AS09]
  - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
  - Lazy data structures [MMZ+01]
    - Watched literals [MMZ+01]
  - Conflict-guided branching [MMZ+01]
    - Lightweight branching heuristics [PD07]
    - Phase saving [PD07]
  - ...

Outline

Clause Learning, UIPs & Minimization

Search Restarts

Lazy Data Structures

Why CDCL Works?

Incremental SAT

Introducing PySAT
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ø</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z → a → ⊥</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b</td>
<td></td>
</tr>
</tbody>
</table>

- Analyze conflict [MS95, MSS96a, MSS96b, MSS99]
- Reasons: x and z
- Decision variable & literals assigned at decision levels less than current
- Create new clause: (¬x ∨ ¬z)
- Can relate clause learning with resolution
- Learned clauses result from (selected) resolution operations
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

- Analyze conflict

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\bot$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b$</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable & literals assigned at decision levels less than current

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\perp$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b$</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable & literals assigned at decision levels less than current
  - Create new clause: $(\overline{x} \lor \overline{z})$

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: x and z
  - Decision variable & literals assigned at decision levels less than current
  - Create new clause: \((\bar{x} \lor \bar{z})\)
- Can relate clause learning with resolution

\[(\bar{a} \lor \bar{b}) \quad (\bar{z} \lor b) \quad (\bar{x} \lor \bar{z} \lor a)\]
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: \((\overline{x} \lor \overline{z})\)

- Can relate **clause learning** with resolution

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning

Level | Dec. | Unit Prop.
--- | --- | ---
0 | $\emptyset$ | |
1 | $x$ | |
2 | $y$ | |
3 | $z$ | $\bot$

- Analyze conflict
  - Reasons: $x$ and $z$
    - Decision variable & literals assigned at decision levels less than current
  - Create new clause: $(\bar{x} \lor \bar{z})$
- Can relate clause learning with resolution

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create new clause: (\(\overline{x} \lor \overline{z}\))

- Can relate clause learning with resolution
  - Learned clauses result from (selected) resolution operations

\[\text{reasons: } x \text{ and } z\]

\[\text{decision variable & literals assigned at decision levels less than current}\]

\[\text{create new clause: } (\overline{x} \lor \overline{z})\]

\[\text{can relate clause learning with resolution}\]

\[\text{learned clauses result from (selected) resolution operations}\]

[MS95, MSS96a, MSS96a, MSS96b, MSS99]
Clause learning – after backtracking

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$z$</td>
<td>$a$ $\perp$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$b$ $\perp$</td>
</tr>
</tbody>
</table>

Clause $(\overline{x} \lor \overline{z})$ is asserting at decision level 1.

Learned clauses are asserting (with exceptions).

Backtracking differs from plain DPLL:

– Always backtrack after a conflict

[MS95, MSS96b, MSS99]
Clause learning – after backtracking

- Clause \((\overline{x} \lor \overline{z})\) is asserting at decision level 1
Clause learning – after backtracking

- Clause \((\overline{x} \lor \overline{z})\) is asserting at decision level 1
Clause learning – after backtracking

- Clause \((\overline{x} \lor \overline{z})\) is asserting at decision level 1
- Learned clauses are asserting (with exceptions) \[\text{[MS95, MSS96b, MSS99]}\]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict \[\text{[MMZ}^+01]\]
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c2</td>
</tr>
<tr>
<td>5</td>
<td>c</td>
<td>c3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c5</td>
</tr>
<tr>
<td>6</td>
<td>f</td>
<td>c6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⊥</td>
</tr>
</tbody>
</table>

Diagram:

- Level 0: ∅
- Level 1: h
- Level 2: b
- Level 3: y
- Level 4: a → c → e → f → g → ⊥

Edges:
- c1, c2, c3, c4, c5, c6
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>↓</td>
<td>c_6</td>
<td>∅</td>
<td>{}</td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c_1</td>
<td>c_4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td>c_1</td>
<td>c_4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>c_1</td>
<td>c_2, c_3</td>
<td>c_4, c_6</td>
<td>{f, g}</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis

Level | Dec. | Unit Prop. |
--- | --- | --- |
0 | ∅ |  
1 | h |  
2 | b |  
3 | y |  
4 | a |  

Step | Var Queue | Extract Var | Antecedent | Recorded Lits | Added to Queue |
--- | --- | --- | --- | --- | --- |
0 | – | ⊥ | c₆ | ∅ | {f, g} |
1 | [f, g] | f | c₄ | {h} | {e} |
### Quiz – conflict analysis

**Level**

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{¬h}</td>
<td>{c, d}</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{¬h}</td>
<td>{c, d}</td>
</tr>
<tr>
<td>4</td>
<td>[c, d]</td>
<td>c</td>
<td>c₁</td>
<td>{¬h, ¬b}</td>
<td>{a}</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>⊥</td>
<td>c₆</td>
<td>{}</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>{}</td>
</tr>
<tr>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{¬h}</td>
<td>{c, d}</td>
</tr>
<tr>
<td>4</td>
<td>[c, d]</td>
<td>c</td>
<td>c₁</td>
<td>{¬h, ¬b}</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>[d, a]</td>
<td>d</td>
<td>c₂</td>
<td>{¬h, ¬b}</td>
<td>{}</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{h}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{h}</td>
<td>{c, d}</td>
</tr>
<tr>
<td>4</td>
<td>[c, d]</td>
<td>c</td>
<td>c₁</td>
<td>{h, b}</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>[d, a]</td>
<td>d</td>
<td>c₂</td>
<td>{h, b}</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>[a]</td>
<td>a</td>
<td>dec var</td>
<td>{h, b, ̅a}</td>
<td>–</td>
</tr>
</tbody>
</table>
### Quiz – conflict analysis

#### Level Dec. Unit Prop.

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

#### Step Var Queue Extract Var Antecedent Recorded Lits Added to Queue

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{¬h}</td>
<td>{c, d}</td>
</tr>
<tr>
<td>4</td>
<td>[c, d]</td>
<td>c</td>
<td>c₁</td>
<td>{¬h, ¬b}</td>
<td>{a}</td>
</tr>
<tr>
<td>5</td>
<td>[d, a]</td>
<td>d</td>
<td>c₂</td>
<td>{¬h, ¬b}</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>[a]</td>
<td>a</td>
<td>dec var</td>
<td>{¬h, ¬b, ¬a}</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>{¬h, ¬b, ¬a}</td>
<td>–</td>
</tr>
</tbody>
</table>
Unique implication points (UIPs)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>a → c → ⊥</td>
</tr>
</tbody>
</table>

Diagram showing the relationships between the unique implication points (UIPs) across different levels.
Unique implication points (UIPs)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

- Learn clause \((\bar{w} \lor \bar{x} \lor \bar{y} \lor \bar{z})\)

\[(\bar{b} \lor \bar{c}) \quad (\bar{w} \lor \bar{a} \lor c) \quad (\bar{x} \lor \bar{a} \lor b) \quad (\bar{y} \lor \bar{z} \lor a)\]

But \(a\) is an UIP \([\text{MS95, MSS99}]\)

– Dominator in DAG for decision level 4
Unique implication points (UIPs)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>

\[(\overline{b} \lor \overline{c}) (\overline{w} \lor \overline{a} \lor \overline{c}) (\overline{x} \lor \overline{a} \lor \overline{b}) (\overline{y} \lor \overline{z} \lor \overline{a})\]

\[(\overline{w} \lor \overline{a} \lor \overline{b})
(\overline{w} \lor \overline{a} \lor \overline{b})
(\overline{w} \lor \overline{x} \lor \overline{a})
(\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z})\]

- Learn clause \((\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z})\)
- But \(a\) is an UIP
  - Dominator in DAG for decision level 4

[MS95, MSS99]
Unique implication points (UIPs)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$w$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$z$</td>
<td>$a$, $c$</td>
</tr>
</tbody>
</table>

- **Learn clause** $(\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z})$
- **But** $a$ is an UIP
  - Dominator in DAG for level 4
- **Learn clause** $(\overline{w} \lor \overline{x} \lor \overline{a})$

[MS95, MSS99]
### Multiple UIPs

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$w$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$x$</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>$y$</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>$z$</td>
<td>$r \rightarrow a \rightarrow c$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$s \rightarrow b \rightarrow \perp$</td>
</tr>
</tbody>
</table>

**First UIP:**
- Learn clause ($\overline{w} \lor \overline{y} \lor \overline{a}$)

**Second UIP:**
- Learn clause ($\overline{x} \lor \overline{z} \lor a$)
  - Clause is not asserting

In practice, smaller clauses are more effective.

- Compare with ($\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z}$)

Multiple UIPs proposed in GRASP [MS95, MSS99].

- First UIP learning proposed in Chaff [MMZ + 01].

- Not used in recent state of the art CDCL SAT solvers.

- Recent results show it can be beneficial on some instances [SSS12].
Multiple UIPs

Level | Dec. | Unit Prop.
---|---|---
0 | ∅ | |
1 | w | |
2 | x | |
3 | y | |
4 | z | r | a | c
   | s | b | ⊥

- **First UIP:**
  - Learn clause \((\bar{w} \lor \bar{y} \lor \bar{a})\)

- Multiple UIPs proposed in GRASP \([MS95, MSS99]\)
- First UIP learning proposed in Chaff \([MMZ+01]\)
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on some instances \([SSS12]\)
Multiple UIPs

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>r</td>
</tr>
</tbody>
</table>

- **First UIP:**
  - Learn clause \((\overline{w} \lor \overline{y} \lor \overline{a})\)

- But there can be more than 1 UIP

**Multiple UIPs proposed in GRASP** \([MS95, MSS99]\)

- First UIP learning proposed in Chaff \([MMZ+01]\)

- Not used in recent state of the art CDCL SAT solvers

- Recent results show it can be beneficial on some instances \([SSS12]\)
Multiple UIPs

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td></td>
</tr>
</tbody>
</table>

- **First UIP:**
  - Learn clause \((\bar{w} \lor \bar{y} \lor \bar{a})\)
- But there can be more than 1 UIP
- **Second UIP:**
  - Learn clause \((\bar{x} \lor \bar{z} \lor a)\)
  - Clause is not asserting

Multiple UIPs proposed in GRASP [MS95, MSS99]

- First UIP learning proposed in Chaff [MMZ 01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on some instances [SSS12]
Multiple UIPs

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>r</td>
</tr>
</tbody>
</table>

- **First UIP:**
  - Learn clause \((\bar{w} \lor \bar{y} \lor \bar{a})\)
  - But there can be more than 1 UIP
- **Second UIP:**
  - Learn clause \((\bar{x} \lor \bar{z} \lor a)\)
  - Clause is not asserting
- **In practice smaller clauses more effective**
  - Compare with \((\bar{w} \lor \bar{x} \lor \bar{y} \lor \bar{z})\)
Multiple UIPs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td>w</td>
<td>2</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td>x</td>
<td>3</td>
<td>y</td>
<td>z</td>
</tr>
<tr>
<td>4</td>
<td>z</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **First UIP:**
  - Learn clause \((\overline{w} \lor \overline{y} \lor \overline{a})\)
- But there can be more than 1 UIP
- **Second UIP:**
  - Learn clause \((\overline{x} \lor \overline{z} \lor a)\)
  - Clause is not asserting
- In practice smaller clauses more effective
  - Compare with \((\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z})\)

- Multiple UIPs proposed in GRASP
  - First UIP learning proposed in Chaff
- Not used in recent state of the art CDCL SAT solvers

[MS95, MSS99]

[MMZ\textsuperscript{+}01]
Multiple UIPs

• First UIP:
  – Learn clause \((\overline{w} \lor \overline{y} \lor \overline{a})\)

• But there can be more than 1 UIP

• Second UIP:
  – Learn clause \((\overline{x} \lor \overline{z} \lor a)\)
  – Clause is not asserting

• In practice smaller clauses more effective
  – Compare with \((\overline{w} \lor \overline{x} \lor \overline{y} \lor \overline{z})\)

• Multiple UIPs proposed in GRASP
  [MS95, MSS99]

• First UIP learning proposed in Chaff
  [MMZ⁺01]

• Not used in recent state of the art CDCL SAT solvers

• Recent results show it can be beneficial on some instances
  [SSS12]
Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c₁, c₄</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td>c₁, c₃, c₄, c₅</td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td>c₁, c₂, c₃</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Level 0: ∅
- Level 1: h
- Level 2: b
- Level 3: y
- Level 4: a

Diagram arrows represent the dependencies:
- c1 from a to c
- c2 from a to d
- c3 from c to e
- c4 from e to f
- c5 from e to g
- c6 from f to ⊥
- c4 from e to f
- c5 from e to g
- c6 from f to ⊥

Table:

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c6</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c4</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
</tbody>
</table>
## Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{h}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
<td>0</td>
<td>–</td>
<td>⊥</td>
<td>c₆</td>
<td>∅</td>
<td>{f, g}</td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
<td>1</td>
<td>[f, g]</td>
<td>f</td>
<td>c₄</td>
<td>{¬h}</td>
<td>{e}</td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>c₁</td>
<td>2</td>
<td>[g, e]</td>
<td>g</td>
<td>c₅</td>
<td>{¬h}</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td>c₄</td>
<td>3</td>
<td>[e]</td>
<td>e</td>
<td>c₃</td>
<td>{¬h, ¬e}</td>
<td>∅</td>
</tr>
</tbody>
</table>
Quiz – conflict analysis with UIP(s)

<table>
<thead>
<tr>
<th>Step</th>
<th>Var Queue</th>
<th>Extract Var</th>
<th>Antecedent</th>
<th>Recorded Lits</th>
<th>Added to Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>–</td>
<td>$\bot$</td>
<td>$c_6$</td>
<td>$\emptyset$</td>
<td>${f, g}$</td>
</tr>
<tr>
<td>1</td>
<td>$[f, g]$</td>
<td>$f$</td>
<td>$c_4$</td>
<td>${\bar{h}}$</td>
<td>${e}$</td>
</tr>
<tr>
<td>2</td>
<td>$[g, e]$</td>
<td>$g$</td>
<td>$c_5$</td>
<td>${\bar{h}}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>$[e]$</td>
<td>$e$</td>
<td>$c_3$</td>
<td>${\bar{h}, \bar{e}}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>6</td>
<td>$[]$</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>${\bar{h}, \bar{e}}$</td>
</tr>
</tbody>
</table>
Quiz (Cont.) – non-chronological backtracking

Without UIP:

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

With UIP:

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>h</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>
Clause minimization I

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>c ⊥</td>
</tr>
</tbody>
</table>

sequences: {x, y, z, a, b, c}
### Clause minimization I

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>c</td>
</tr>
</tbody>
</table>

- **Learn clause** \((\overline{x} \lor \overline{y} \lor \overline{z} \lor \overline{b})\)

\[
\begin{align*}
(\overline{a} \lor \overline{c}) & \quad (\overline{z} \lor \overline{b} \lor c) & \quad (\overline{x} \lor \overline{y} \lor \overline{z} \lor a) \\
(\overline{z} \lor \overline{b} \lor \overline{a}) & \quad (\overline{x} \lor \overline{y} \lor \overline{z} \lor \overline{b})
\end{align*}
\]
Clause minimization I

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>c</td>
</tr>
</tbody>
</table>

- Learn clause $(\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})$
- Apply self-subsuming resolution (i.e. **local minimization**) [SB09]

\[(\bar{a} \lor \bar{c}) \quad (\bar{z} \lor \bar{b} \lor c) \quad (\bar{x} \lor \bar{y} \lor \bar{z} \lor a) \quad (\bar{x} \lor b)\]

\[(\bar{z} \lor \bar{b} \lor \bar{a}) \quad (\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})\]
Clause minimization I

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>c</td>
</tr>
</tbody>
</table>

- Learn clause \((\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})\)
- Apply self-subsuming resolution (i.e. local minimization) [SB09]
Clause minimization I

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>z</td>
<td>c</td>
</tr>
</tbody>
</table>

- Learn clause \((\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})\)
- Apply self-subsuming resolution (i.e. local minimization) [SB09]
- Learn clause \((\bar{x} \lor \bar{y} \lor \bar{z})\)
Clause minimization II

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>\∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(w)</td>
<td>(a) (c)</td>
</tr>
<tr>
<td>2</td>
<td>(x)</td>
<td>(e) (\bot)</td>
</tr>
</tbody>
</table>

Cannot apply self-subsuming resolution – Resolving with reason of \(c\) yields \((\overline{w} \lor \overline{x} \lor \overline{a} \lor \overline{b})\).

Can apply recursive minimization.

Learn clause \((\overline{w} \lor \overline{x})\).

Marked nodes: literals in learned clause \([SB09]\).

Trace back from \(c\) until marked nodes or new decision nodes – Drop literal \(c\) if only marked nodes visited.

Recursive minimization runs in (amortized) linear time.
Clause minimization II

- Learn clause \((\bar{w} \lor \bar{x} \lor \bar{c})\)
Clause minimization II

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Ø</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⊥</td>
</tr>
</tbody>
</table>

- Learn clause \((\overline{w} \lor \overline{x} \lor \overline{c})\)
- **Cannot** apply self-subsuming resolution
  - Resolving with reason of \(c\) yields \((\overline{w} \lor \overline{x} \lor \overline{a} \lor \overline{b})\)
Learn clause \((\lnot w \lor \lnot x \lor \lnot c)\)

- **Cannot** apply self-subsuming resolution
  - Resolving with reason of \(c\) yields \((\lnot w \lor \lnot x \lor \lnot a \lor \lnot b)\)

- Can apply recursive minimization
Clause minimization II

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\emptyset$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$w$</td>
<td>$a$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>$c$</td>
</tr>
<tr>
<td>2</td>
<td>$x$</td>
<td>$e$</td>
</tr>
<tr>
<td></td>
<td>$d$</td>
<td>$\bot$</td>
</tr>
</tbody>
</table>

- **Learn clause** $(\overline{w} \lor \overline{x} \lor \overline{c})$
- **Cannot** apply self-subsuming resolution
  - Resolving with reason of $c$ yields $(\overline{w} \lor \overline{x} \lor \overline{a} \lor \overline{b})$
- **Can apply** recursive minimization

- **Marked nodes**: literals in learned clause

[SB09]
**Clause minimization II**

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⊥</td>
</tr>
</tbody>
</table>

- **Learn clause** \((\overline{w} \lor \overline{x} \lor \overline{c})\)
- **Cannot** apply self-subsuming resolution
  - Resolving with reason of \(c\) yields \((\overline{w} \lor \overline{x} \lor \overline{a} \lor \overline{b})\)
- **Can apply** recursive minimization

- **Marked nodes:** literals in learned clause
- **Trace back from** \(c\) until **marked** nodes or new decision nodes
  - Drop literal \(c\) if only **marked** nodes visited

[SB09]
Clause minimization II

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>w</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>c</td>
</tr>
<tr>
<td>2</td>
<td>x</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
</tbody>
</table>

- Learn clause $(\overline{w} \lor \overline{x} \lor \overline{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of $c$ yields $(\overline{w} \lor \overline{x} \lor \overline{a} \lor \overline{b})$
- Can apply recursive minimization
- Learn clause $(\overline{w} \lor \overline{x})$

- Marked nodes: literals in learned clause
- Trace back from $c$ until marked nodes or new decision nodes
  - Drop literal $c$ if only marked nodes visited

[SB09]
Clause minimization II

- Learn clause \((\bar{w} \lor \bar{x} \lor \bar{c})\)
- Cannot apply self-subsuming resolution
  - Resolving with reason of \(c\) yields \((\bar{w} \lor \bar{x} \lor \bar{a} \lor \bar{b})\)
- Can apply recursive minimization
- Learn clause \((\bar{w} \lor \bar{x})\)

- **Marked nodes**: literals in learned clause
- Trace back from \(c\) until marked nodes or new decision nodes
  - Drop literal \(c\) if only marked nodes visited
- Recursive minimization runs in (amortized) linear time

[SB09]
Quiz – conflict clause minimization

Level | Dec. | Unit Prop. |
--- | --- | --- |
0 | ∅ | |
1 | a | |
2 | b → r → d → s → g | |
3 | y | |
4 | c → e → h → ⊥ | |

Learned clause: \((a ∨ r ∨ c ∨ d ∨ g)\)
Minimized clause: \((a ∨ r ∨ c ∨ d ∨ g)\)
Quiz – conflict clause minimization

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⊥</td>
</tr>
</tbody>
</table>

Learned clause: $(\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})$

Minimized clause: $(\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})$
Quiz – conflict clause minimization

Target | Curr Var | Marked | Unmarked | Vars to Trace | Action

Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)
Quiz – conflict clause minimization

Level  | Dec. | Unit Prop.
-------|------|-------------
0      | $\emptyset$ | |
1      | $a$ | |
2      | $b \rightarrow r \rightarrow d \rightarrow s \rightarrow g$ | |
3      | $y$ | |
4      | $c \rightarrow e \rightarrow h \rightarrow \bot$ | |

Learned clause: $\left( \bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g} \right)$$
Minimized clause: $\left( \bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g} \right)$

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>$g$</td>
<td>${a, d, r, c}$</td>
<td>$\emptyset$</td>
<td>$[s]$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization

Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g)</td>
<td>(g)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([s])</td>
<td>–</td>
</tr>
<tr>
<td>(g)</td>
<td>(s)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([d])</td>
<td>–</td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization

Learned clause: \((\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})\)

Minimized clause: \((\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g)</td>
<td>(g)</td>
<td>{a, d, r, c}</td>
<td>(\emptyset)</td>
<td>[s]</td>
<td>–</td>
</tr>
<tr>
<td>(g)</td>
<td>(s)</td>
<td>{a, d, r, c}</td>
<td>(\emptyset)</td>
<td>[d]</td>
<td>–</td>
</tr>
<tr>
<td>(g)</td>
<td>(d)</td>
<td>{a, d, r, c}</td>
<td>(\emptyset)</td>
<td>[]</td>
<td>(d) marked, skip</td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e</td>
</tr>
</tbody>
</table>

Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>g</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td>[s]</td>
<td>–</td>
</tr>
<tr>
<td>g</td>
<td>s</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td>[d]</td>
<td>–</td>
</tr>
<tr>
<td>g</td>
<td>d</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td></td>
<td>(d) marked, skip</td>
</tr>
<tr>
<td>g</td>
<td>–</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td></td>
<td>no unmarked vars; (\therefore) drop (g)</td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization

Learned clause: \((\neg a \lor \neg r \lor \neg c \lor \neg d \lor \neg g)\)

Minimized clause: \((\neg a \lor \neg r \lor \neg c \lor \neg d)\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g)</td>
<td>(g)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([s])</td>
<td>(\cdot)</td>
</tr>
<tr>
<td>(g)</td>
<td>(s)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([d])</td>
<td>(\cdot)</td>
</tr>
<tr>
<td>(g)</td>
<td>(d)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(d) marked, skip</td>
</tr>
<tr>
<td>(g)</td>
<td>(\cdot)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\cdot) drop (g)</td>
</tr>
<tr>
<td>(d)</td>
<td>(d)</td>
<td>({a, r, c})</td>
<td>(\emptyset)</td>
<td>([r])</td>
<td>(\cdot)</td>
</tr>
</tbody>
</table>
## Quiz – conflict clause minimization

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>h</td>
</tr>
<tr>
<td></td>
<td></td>
<td>⊥</td>
</tr>
</tbody>
</table>

Learned clause: \((\bar{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\bar{a} \lor \overline{r} \lor \overline{c} \lor \overline{d})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td>[s]</td>
<td>–</td>
</tr>
<tr>
<td>g</td>
<td>g</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td>[d]</td>
<td>–</td>
</tr>
<tr>
<td>g</td>
<td>s</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td></td>
<td>d marked, skip</td>
</tr>
<tr>
<td>g</td>
<td>d</td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td></td>
<td>no unmarked vars; \therefore drop g</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{a, d, r, c}</td>
<td>∅</td>
<td>[r]</td>
<td>–</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>{a, r, c}</td>
<td>∅</td>
<td></td>
<td>r marked, skip</td>
</tr>
<tr>
<td>d</td>
<td>r</td>
<td>{a, r, c}</td>
<td>∅</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization

Learned clause: \((\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})\)

Minimized clause: \((\bar{a} \lor \bar{r} \lor \bar{c})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>(g)</td>
<td>(g)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([s])</td>
<td>–</td>
</tr>
<tr>
<td>(g)</td>
<td>(s)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>([d])</td>
<td>–</td>
</tr>
<tr>
<td>(g)</td>
<td>(d)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>(_)</td>
<td>(d) marked, skip</td>
</tr>
<tr>
<td>(g)</td>
<td>(–)</td>
<td>({a, d, r, c})</td>
<td>(\emptyset)</td>
<td>(_)</td>
<td>no unmarked vars; (\therefore) drop (g)</td>
</tr>
<tr>
<td>(d)</td>
<td>(d)</td>
<td>({a, r, c})</td>
<td>(\emptyset)</td>
<td>([r])</td>
<td>–</td>
</tr>
<tr>
<td>(d)</td>
<td>(r)</td>
<td>({a, r, c})</td>
<td>(\emptyset)</td>
<td>(_)</td>
<td>(r) marked, skip</td>
</tr>
<tr>
<td>(d)</td>
<td>(–)</td>
<td>({a, r, c})</td>
<td>(\emptyset)</td>
<td>(_)</td>
<td>no unmarked vars; (\therefore) drop (d)</td>
</tr>
</tbody>
</table>
Quiz – conflict clause minimization (cont.)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e</td>
</tr>
</tbody>
</table>

Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c})\)
Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c})\)
Quiz – conflict clause minimization (cont.)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e</td>
</tr>
</tbody>
</table>

Learned clause: \((\overline{a} \lor \overline{r} \lor \overline{c} \lor \overline{d} \lor \overline{g})\)

Minimized clause: \((\overline{a} \lor \overline{r} \lor \overline{c})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>{a, c}</td>
<td>∅</td>
<td>[a, b]</td>
<td>–</td>
</tr>
<tr>
<td>r</td>
<td>a</td>
<td>{a, c}</td>
<td>∅</td>
<td>[b]</td>
<td>a marked</td>
</tr>
<tr>
<td>r</td>
<td>b</td>
<td>{a, c}</td>
<td>{b}</td>
<td>[]</td>
<td>(b) decision &amp; unmarked</td>
</tr>
<tr>
<td>r</td>
<td>–</td>
<td>{a, c}</td>
<td>{b}</td>
<td>[]</td>
<td>unmarked vars; (\therefore) keep (r)</td>
</tr>
</tbody>
</table>
### Quiz – conflict clause minimization (cont.)

<table>
<thead>
<tr>
<th>Level</th>
<th>Dec.</th>
<th>Unit Prop.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>∅</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td>r → d → s → g</td>
</tr>
<tr>
<td>3</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>e → h → ⊥</td>
</tr>
</tbody>
</table>

#### Learned clause:

\((\bar{a} \lor \bar{r} \lor \bar{c} \lor \bar{d} \lor \bar{g})\)

#### Minimized clause:

\((\bar{a} \lor \bar{r} \lor \bar{c})\)

<table>
<thead>
<tr>
<th>Target</th>
<th>Curr Var</th>
<th>Marked</th>
<th>Unmarked</th>
<th>Vars to Trace</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>r</td>
<td>{a, c}</td>
<td>∅</td>
<td>[a, b]</td>
<td>–</td>
</tr>
<tr>
<td>r</td>
<td>a</td>
<td>{a, c}</td>
<td>∅</td>
<td>[b]</td>
<td>a marked</td>
</tr>
<tr>
<td>r</td>
<td>b</td>
<td>{a, c}</td>
<td>{b}</td>
<td></td>
<td>b decision &amp; unmarked</td>
</tr>
<tr>
<td>r</td>
<td>–</td>
<td>{a, c}</td>
<td>{b}</td>
<td></td>
<td>unmarked vars; ∴ keep r</td>
</tr>
<tr>
<td>a, c</td>
<td>–</td>
<td>–</td>
<td>∅</td>
<td></td>
<td>a, c decision variables; keep both</td>
</tr>
</tbody>
</table>
Branch randomization

- Heavy-tail behavior:

  - 10000 runs, branching randomization on satisfiable industrial instance

    \[ \therefore \text{use rapid randomized restarts (search restarts)} \]
Search restarts

- Restart search after a number of conflicts

---

Proof complexity arguments

- Clause learning (very) effective in between restarts
Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist

Proof complexity arguments
- Clause learning (very) effective in between restarts.
Search restarts

- Restart search after a number of conflicts
  - Increase \textit{cutoff} after each restart
    - Guarantees completeness
    - Different policies exist
  - Effective for SAT & UNSAT formulas. \textit{Why?}
Search restarts

• Restart search after a number of conflicts
  – Increase \textit{cutoff} after each restart
    ▶ Guarantees completeness
    ▶ Different policies exist
  – Effective for SAT & UNSAT formulas. \textbf{Why?}
    ▶ Proof complexity arguments
Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist
  - Effective for SAT & UNSAT formulas. **Why?**
    - Proof complexity arguments
  - Clause learning (very) effective in between restarts
Outline

Clause Learning, UIPs & Minimization

Search Restarts

Lazy Data Structures

Why CDCL Works?

Incremental SAT

Introducing PySAT
Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal \( l \) should access clauses containing \( l \) and \( \overline{l} \)
  - Why?
- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal \( l \) should access clauses containing \( l \) and \( \bar{l} \)
  - Why? Unit propagation
• Recap states of a clause: **unresolved, unit, falsified, satisfied**

• Each literal \( l \) should access clauses containing \( l \) and \( \overline{l} \)
  – Why? Unit propagation

• Clause with \( k \) literals results in \( k \) references, from literals to the clause
Data structures basics

• Recap states of a clause: unresolved, unit, falsified, satisfied
• Each literal $l$ should access clauses containing $l$ and $\overline{l}$
  – Why? Unit propagation
• Clause with $k$ literals results in $k$ references, from literals to the clause
• Number of clause references equals number of literals, $L$
Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal $l$ should access clauses containing $l$ and $\overline{l}$
  - Why? Unit propagation

- Clause with $k$ literals results in $k$ references, from literals to the clause

- Number of clause references equals number of literals, $L$
  - Clause learning can generate large clauses
    - Worst-case size: $O(n)$
Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal \( l \) should access clauses containing \( l \) and \( \overline{l} \)
  - Why? Unit propagation
- Clause with \( k \) literals results in \( k \) references, from literals to the clause
- Number of clause references equals number of literals, \( L \)
  - Clause learning can generate large clauses
    - Worst-case size: \( O(n) \)
  - Worst-case number of literals: \( O(mn) \)
Data structures basics

• Recap states of a clause: unresolved, unit, falsified, satisfied
• Each literal \( l \) should access clauses containing \( l \) and \( \bar{l} \)
  – Why? Unit propagation

• Clause with \( k \) literals results in \( k \) references, from literals to the clause
• Number of clause references equals number of literals, \( L \)
  – Clause learning can generate large clauses
    – Worst-case size: \( O(n) \)
  – Worst-case number of literals: \( O(m n) \)
  – In practice,

  Unit propagation slow-down worse than linear as clauses are learned
Data structures basics

- Recap states of a clause: **unresolved**, **unit**, **falsified**, **satisfied**
- Each literal \( l \) should access clauses containing \( l \) and \( \overline{l} \)
  - Why? Unit propagation
- Clause with \( k \) literals results in \( k \) references, from literals to the clause
- Number of clause references **equals** number of literals, \( L \)
  - Clause learning can generate large clauses
    - Worst-case size: \( \mathcal{O}(n) \)
    - Worst-case number of literals: \( \mathcal{O}(mn) \)
    - In practice, Unit propagation slow-down worse than linear as clauses are learned!
- Clause learning to be effective requires a more efficient representation:
Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal $l$ should access clauses containing $l$ and $\overline{l}$
  - Why? Unit propagation
- Clause with $k$ literals results in $k$ references, from literals to the clause
- Number of clause references equals number of literals, $L$
  - Clause learning can generate large clauses
    - Worst-case size: $O(n)$
  - Worst-case number of literals: $O(mn)$
  - In practice,
    Unit propagation slow-down worse than linear as clauses are learned
- Clause learning to be effective requires a more efficient representation: **Watched Literals**

[MMZ^+01]
Data structures basics

• Recap states of a clause: unresolved, unit, falsified, satisfied
• Each literal $l$ should access clauses containing $l$ and $\bar{l}$
  – Why? Unit propagation
• Clause with $k$ literals results in $k$ references, from literals to the clause
• Number of clause references equals number of literals, $L$
  – Clause learning can generate large clauses
    ▶ Worst-case size: $O(n)$
  – Worst-case number of literals: $O(mn)$
  – In practice,
    Unit propagation slow-down worse than linear as clauses are learned!
• Clause learning to be effective requires a more efficient representation: Watched Literals
  – Watched literals are one example of lazy data structures
    ▶ But there are others
Watched literals

At DLevel 2: clause is unresolved

At DLevel 3: watch updated

At DLevel 4: watch updated

At DLevel 5: clause is unit

Literal D assigned value 1; clause becomes satisfied

After backtracking to DLevel 1

Watched literals untouched
Watched literals

Watch 2 unassigned literals in each clause
Watched literals

Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved
Watched literals

Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

At DLevel 3: watch updated
Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

At DLevel 3: watch updated

At DLevel 4: watch updated
Watched literals

Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

At DLevel 3: watch updated

At DLevel 4: watch updated

At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied
Watched literals

Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

At DLevel 3: watch updated

At DLevel 4: watch updated

At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied

After backtracking to DLevel 1
Watched literals untouched
Watched literals – different implementations exist!

Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

At DLevel 3: watch updated

At DLevel 4: watch updated

At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied

After backtracking to DLevel 1
Watched literals untouched
Additional key techniques

- **Conflict-driven branching** [MMZ⁺01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
  - Recent promising ML-based branching [LGPC16a, LGPC16b]

- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete larger clauses [MSS96b, MSS99]
  - Delete less used clauses [GN02, ES03]
  - Delete based on LBD metric [AS09]

- Other effective techniques:
  - Phase saving [PD07]
  - Novel restart strategies [Hua07, BF15, LOM +18]
  - Preprocessing/inprocessing [JHB12, HJL +15]
  - Clause minimization: LBD-based and UP-based [AS09, LLX +17]
Additional key techniques

• **Conflict-driven branching**
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
  - Recent promising ML-based branching
    
  [MMZ\(^+\)01]

• **Clause deletion policies**
  - Not practical to keep all learned clauses
  - Delete larger clauses
  - Delete less used clauses
  - Delete based on LBD metric

 [MSS96b, MSS99]
 [GN02, ES03]
 [AS09]
### Additional key techniques

- **Conflict-driven branching**  
  - Use conflict to bias variables to branch on, associate score with each variable  
  - Prefer recent bias by regularly decreasing variable scores  
  - Recent promising ML-based branching  
  [LGPC16a, LGPC16b]

- **Clause deletion policies**  
  - Not practical to keep all learned clauses  
  - Delete larger clauses  
  [MSS96b, MSS99]  
  - Delete less used clauses  
  [GN02, ES03]  
  - Delete based on LBD metric  
  [AS09]

- **Other effective techniques:**  
  - Phase saving  
  [PD07]  
  - Novel restart strategies  
  [Hua07, BF15, LOM+18]  
  - Preprocessing/inprocessing  
  [JHB12, HJL+15]  
  - Clause minimization: LBD-based and UP-based  
  [AS09, LLX+17]
Outline

Clause Learning, UIPs & Minimization

Search Restarts

Lazy Data Structures

Why CDCL Works?

Incremental SAT

Introducing PySAT
Why CDCL works – a practitioner’s view

- **GRASP-like clause learning** extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit’s internal structure
    - Several ideas originated in earlier work

  [MSS93, MSS94]

- Understanding problem structure is essential
  - Clauses are learned locally to each decision level
  - UIPs further localize the learned clauses
  - GRASP-like clause learning aims at learning small clauses, related with the sources of conflicts
  - Most practical problem instances exhibit the structure GRASP-like clause learning is most effective on

  [Stu13]

- There are also proof complexity arguments
  - [BKS04, PD09, PD11]
Why CDCL works – a practitioner’s view

- **GRASP-like clause learning** extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit’s internal structure
    - Several ideas originated in earlier work [MSS93, MSS94]

- Understanding problem structure is **essential**
  - Clauses are learned locally to each decision level
  - UIPs further localize the learned clauses
  - GRASP-like clause learning aims at learning small clauses, related with the sources of conflicts
  - Most practical problem instances exhibit the structure GRASP-like clause learning is most effective on
    - Most problems are **not** natively represented in clausal form [Stu13]
Why CDCL works – a practitioner’s view

- **GRASP-like clause learning** extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit’s internal structure
    - Several ideas originated in earlier work [MSS93, MSS94]

- Understanding problem **structure is essential**
  - Clauses are learned **locally** to each decision level
  - UIPs further **localize** the learned clauses
  - GRASP-like clause learning aims at learning **small** clauses, related with the sources of conflicts
  - Most practical problem instances exhibit the **structure** GRASP-like clause learning is most effective on
    - Most problems are **not** natively represented in clausal form [Stu13]

- There are also proof complexity arguments [BKS04, PD09, PD11]
Outline

Clause Learning, UIPs & Minimization

Search Restarts

Lazy Data Structures

Why CDCL Works?

Incremental SAT

Introducing PySAT
Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes & remember already learning clauses (that still apply)

\[ c_i \lor s_i \]
- To activate clause: add assumption $s_i = 1$
- To deactivate clause: add assumption $s_i = 0$ (optional)
- To remove clause: add unit ($s_i$)

Any learned clause contains explanation given working assumptions (more next)
Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes \& remember already learning clauses (that still apply)
- Most often used solution: [ES03]
Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes & remember already learning clauses (that still apply)
- Most often used solution:
  - Use activation/selector/indicator variables

<table>
<thead>
<tr>
<th>Given clause</th>
<th>Added to SAT solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>$c_i \lor \overline{s_i}$</td>
</tr>
</tbody>
</table>
Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes & remember already learning clauses (that still apply)
- Most often used solution:
  - Use activation(selector/indicator) variables
    - Given clause  |  Added to SAT solver
      \[ c_i \]  |  \[ c_i \lor \overline{s_i} \]
    - To activate clause: add assumption \[ s_i = 1 \]

[ES03]
Incremental SAT solving

- SAT solver often called multiple times on related formulas

- It helps to make incremental changes & remember already learning clauses (that still apply)

- Most often used solution:
  - Use activation/selector/indicator variables

<table>
<thead>
<tr>
<th>Given clause</th>
<th>Added to SAT solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>$c_i \lor \overline{s_i}$</td>
</tr>
</tbody>
</table>

- To activate clause: add assumption $s_i = 1$
- To deactivate clause: add assumption $s_i = 0$ (optional)
Incremental SAT solving

• SAT solver often called multiple times on related formulas

• It helps to make incremental changes \& remember already learning clauses (that still apply)

• Most often used solution:
  - Use activation/selector/indicator variables

<table>
<thead>
<tr>
<th>Given clause</th>
<th>Added to SAT solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_i )</td>
<td>( c_i \lor \overline{s_i} )</td>
</tr>
</tbody>
</table>

  - To activate clause: add assumption \( s_i = 1 \)
  - To deactivate clause: add assumption \( s_i = 0 \) (optional)
  - To remove clause: add unit \( (\overline{s_i}) \)
Incremental SAT solving

- SAT solver often called multiple times on related formulas

- It helps to make incremental changes & remember already learning clauses (that still apply)

- Most often used solution:
  - Use activation/selector/indicator variables

<table>
<thead>
<tr>
<th>Given clause</th>
<th>Added to SAT solver</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_i$</td>
<td>$c_i \lor \overline{s_i}$</td>
</tr>
</tbody>
</table>

- To activate clause: add assumption $s_i = 1$
- To deactivate clause: add assumption $s_i = 0$ (optional)
- To remove clause: add unit ($\overline{s_i}$)
- Any learned clause contains explanation given working assumptions (more next)
An example

\[ B = \{ (\bar{a} \lor b), (\bar{a} \lor c) \} \]
\[ S = \{ (a \lor s_1), (\bar{b} \lor \bar{c} \lor s_2), (a \lor \bar{c} \lor s_3), (a \lor \bar{b} \lor s_4) \} \]

- Background knowledge \( B \): final clauses, i.e. no indicator variables
- Soft clauses \( S \): add indicator variables \( \{s_1, s_2, s_3, s_4\} \)
An example

\[ B = \{(\bar{a} \lor b), (\bar{a} \lor c)\} \]

\[ S = \{(a \lor \bar{s}_1), (\bar{b} \lor \bar{c} \lor \bar{s}_2), (a \lor \bar{c} \lor \bar{s}_3), (a \lor \bar{b} \lor \bar{s}_4)\} \]

- Background knowledge \( B \): final clauses, i.e. no indicator variables
- Soft clauses \( S \): add indicator variables \( \{s_1, s_2, s_3, s_4\} \)
- E.g. given assumptions \( \{s_1 = 1, s_2 = 0, s_3 = 0, s_4 = 1\} \), SAT solver handles formula:

\[ F = \{(\bar{a} \lor b), (\bar{a} \lor c), (a), (a \lor \bar{b})\} \]

which is satisfiable
Quiz – what happens in this case?

\[ \mathcal{B} = \{(\bar{a} \lor b), (\bar{a} \lor c)\} \]

\[ S = \{(a \lor \bar{s}_1), (\bar{b} \lor \bar{c} \lor \bar{s}_2), (a \lor \bar{c} \lor \bar{s}_3), (a \lor \bar{b} \lor \bar{s}_4)\} \]

- Given assumptions \( \{s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1\} \)?
Quiz – what happens in this case?

\[ B = \{ (\overline{a} \lor b), (\overline{a} \lor c) \} \]

\[ S = \{ (a \lor \overline{s_1}), (\overline{b} \lor \overline{c} \lor \overline{s_2}), (a \lor \overline{c} \lor \overline{s_3}), (a \lor \overline{b} \lor \overline{s_4}) \} \]

- Given assumptions \( \{ s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1 \} \)?

Unsatisfiable core: 1st and 2nd clauses of \( S \), given \( B \)
Quiz – what happens in this case?

\[ \mathcal{B} = \{ (\bar{a} \lor b), (\bar{a} \lor c) \} \]

\[ \mathcal{S} = \{ (a \lor \bar{s}_1), (\bar{b} \lor \bar{c} \lor \bar{s}_2), (a \lor \bar{c} \lor \bar{s}_3), (a \lor \bar{b} \lor \bar{s}_4) \} \]

- Given assumptions \( \{ s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1 \} \)?

- **Unsatisfiable core**: 1\(^{st}\) and 2\(^{nd}\) clauses of \( \mathcal{S} \), given \( \mathcal{B} \).
Outline

Clause Learning, UIPs & Minimization

Search Restarts

Lazy Data Structures

Why CDCL Works?

Incremental SAT

Introducing PySAT
Overview of PySAT

- Open source, available on github
- Comprehensive list of SAT solvers
- Comprehensive list of cardinality encodings
- Fairly comprehensive documentation
- Several use cases
Overview of PySAT

- Open source, available on github

PySAT modules

- solvers module
- formula module
- cardenc module

PySAT API
Overview of PySAT

- Open source, available on github
- Comprehensive list of SAT solvers
- Comprehensive list of cardinality encodings
- Fairly comprehensive documentation
- Several use cases

PySAT API

PySAT modules

cardenc module

solvers module

formula module
### Available solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glucose</td>
<td>3.0</td>
</tr>
<tr>
<td>Glucose</td>
<td>4.1</td>
</tr>
<tr>
<td>Lingeling</td>
<td>bbc-9230380-160707</td>
</tr>
<tr>
<td>Minicard</td>
<td>1.2</td>
</tr>
<tr>
<td>Minisat</td>
<td>2.2 release</td>
</tr>
<tr>
<td>Minisat</td>
<td>GitHub version</td>
</tr>
</tbody>
</table>

- Solvers can either be used *incrementally* or *non-incrementally*.
- Tools can use *multiple solvers*, e.g. for *hitting set dualization* or *CEGAR*-based QBF solving.

**URL:**
## Formula manipulation

### Features

- **CNF & Weighted CNF (WCNF)**
- Read formulas from file/string
- Write formulas to file
- Append clauses to formula
- Negate CNF formulas
- Translate between CNF and WCNF
- ID manager

### URL:

https://pysathq.github.io/docs/html/api/formula.html
Available cardinality encodings

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pairwise</td>
<td>AtMost1</td>
</tr>
<tr>
<td>bitwise</td>
<td>AtMost1</td>
</tr>
<tr>
<td>ladder</td>
<td>AtMost1</td>
</tr>
<tr>
<td>sequential counter</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>sorting network</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>cardinality network</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>totalizer</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>mtotalizer</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>kmtotalizer</td>
<td>AtMost$k$</td>
</tr>
</tbody>
</table>

- Also **AtLeast$k$** and **Equals$k$** constraints

- **URL:**
  https://pysathq.github.io/docs/html/api/card.html
Available cardinality encodings – more later

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>pairwise</td>
<td>AtMost1</td>
</tr>
<tr>
<td>bitwise</td>
<td>AtMost1</td>
</tr>
<tr>
<td>ladder</td>
<td>AtMost1</td>
</tr>
<tr>
<td>sequential counter</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>sorting network</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>cardinality network</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>totalizer</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>mtotalizer</td>
<td>AtMost$k$</td>
</tr>
<tr>
<td>kmtotalizer</td>
<td>AtMost$k$</td>
</tr>
</tbody>
</table>

- Also AtLeast$k$ and Equals$k$ constraints

- **URL:**
  https://pysathq.github.io/docs/html/api/card.html
Installation & info

- **Installation:**
  
  ```
  $ [sudo] pip2|pip3 install python-sat
  ```

- **Website:** [https://pysathq.github.io/](https://pysathq.github.io/)
Basic interface – Python3

```python
>>> from pysat.card import *
>>> am1 = CardEnc.atmost(lits=[1, -2, 3], encoding=EncType.pairwise)
>>> print(am1.clauses)
[[-1, 2], [-1, -3], [2, -3]]

>>> from pysat.solvers import Solver
>>> with Solver(name='m22', bootstrap_with=am1.clauses) as s:
...     if s.solve(assumptions=[1, 2, 3]) == False:
...         print(s.get_core())
[3, 1]
```
Part 2

Problem Modeling for SAT
Quiz – solving Sudoku (first attempt)
Quiz – solving Sudoku (first attempt)
Quiz – solving Sudoku (first attempt)

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>9</td>
<td></td>
<td>7</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

- How to solve Sudoku with constraints / SAT?
A solution in Prolog CLPFD

```
:- use_module(library(clpfd)).

sudoku(Rows) :-
    length(Rows, 9),
    maplist(same_length(Rows), Rows),
    append(Rows, Vs),
    Vs ins 1..9,
    maplist(all_distinct, Rows),
    transpose(Rows, Columns),
    maplist(all_distinct, Columns),
    Rows = [As, Bs, Cs, Ds, Es, Fs, Gs, Hs, Is],
    blocks(As, Bs, Cs),
    blocks(Ds, Es, Fs),
    blocks(Gs, Hs, Is).

blocks([], [], []).
blocks([[N1, N2, N3|Ns1], [N4, N5, N6|Ns2], [N7, N8, N9|Ns3]] :-
    all_distinct([[N1, N2, N3, N4, N5, N6, N7, N8, N9]],
    blocks(Ns1, Ns2, Ns3)).
```
A solution with Minizinc

```minizinc
int: S;
int: N = S * S;
array[1..N,1..N] of var 1..N: puzzle;
include "alldifferent.mzn"

% All cells in a row, in a column, and in a subsquare are different.
constraint
forall(i in 1..N)( alldifferent(j in 1..N)( puzzle[i,j] )) /
forall(j in 1..N)( alldifferent(i in 1..N)( puzzle[i,j] )) /
forall(i,j in 1..S)
  ( alldifferent(p,q in 1..S)( puzzle[S*(i-1)+p,
      S*(j-1)+q] ))

solve satisfy;

output [ "sudoku:\n" ] ++
[ show(puzzle[i,j]) ++
  if j = N then
    if i mod S = 0 /
    i < N then "\n\n" else "\n" endif
  else
    if j mod S = 0 then " " else " " endif
  endif
| i,j in 1..N ];
```
Solving Sudoku – with constraints

• Modeling the problem with integer variables:
  – Rows: \( i = 1, \ldots, 9 \)
  – Columns: \( j = 1, \ldots, 9 \)
  – Variables: \( v_{i,j} \in \{1, 2, \ldots, 9\}, \ i, j \in \{1, \ldots, 9\} \)

• Constraints:
Solving Sudoku – with constraints

- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i, j \in \{1, \ldots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \ldots, 9\}$: \text{alldifferent}(v_{i,1}, \ldots, v_{i,9})
Solving Sudoku – with constraints

- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i,j \in \{1, \ldots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{i,1}, \ldots, v_{i,9})$
  - Each value used exactly once in each column:
    - For $j \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{1,j}, \ldots, v_{9,j})$
Solving Sudoku – with constraints

- Modeling the problem with integer variables:
  - Rows: \( i = 1, \ldots, 9 \)
  - Columns: \( j = 1, \ldots, 9 \)
  - Variables: \( v_{i,j} \in \{1, 2, \ldots, 9\}, \ i,j \in \{1, \ldots, 9\} \)

- Constraints:
  - Each value used exactly once in each row:
    - For \( i \in \{1, \ldots, 9\} \): \( \text{alldifferent}(v_{i,1}, \ldots, v_{i,9}) \)
  - Each value used exactly once in each column:
    - For \( j \in \{1, \ldots, 9\} \): \( \text{alldifferent}(v_{1,j}, \ldots, v_{9,j}) \)
  - Each value used exactly once in each 3 × 3 sub-grid:
    - For \( i,j \in \{0, 1, 2\} \):
      - \( \text{alldifferent}(v_{3i+1,3j+1}, v_{3i+1,3j+2}, v_{3i+1,3j+3}, v_{3i+2,3j+1}, \ldots, v_{3i+3,3j+1}, \ldots) \)
Solving Sudoku – propositional logic – variables

- Modeling with *propositional* variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j,k} \in \{0, 1\}, \ i, j, k \in \{1, \ldots, 9\}$
Solving Sudoku – propositional logic – constraints

- Value in each cell is valid:
  - For \( i, j \in \{1, \ldots, 9\} \):
    \[
    \sum_{k=1}^{9} v_{i,j,k} = 1
    \]

- Each value used exactly once in each row:
  - For \( i \in \{1, \ldots, 9\}, \; k \in \{1, \ldots, 9\} \):
    \[
    \sum_{j=1}^{9} v_{i,j,k} = 1
    \]

- Each value used exactly once in each column:
  - For \( j \in \{1, \ldots, 9\}, \; k \in \{1, \ldots, 9\} \):
    \[
    \sum_{i=1}^{9} v_{i,j,k} = 1
    \]

- Each value used exactly once in each 3 \( \times \) 3 sub-grid:
  - For \( i, j \in \{0, 1, 2\}, \; k \in \{1, \ldots, 9\} \):
    \[
    \sum_{r=1}^{3} \sum_{s=1}^{3} v_{3i+r,3j+s,k} = 1
    \]
• Value in each cell is valid:
  – For $i, j \in \{1, \ldots, 9\}$:
    \[ \sum_{k=1}^{9} v_{i,j,k} = 1 \]

• Each value used exactly once in each row:
  – For $i \in \{1, \ldots, 9\}, k \in \{1, \ldots, 9\}$:
    \[ \sum_{j=1}^{9} v_{i,j,k} = 1 \]

• Each value used exactly once in each column:
  – For $j \in \{1, \ldots, 9\}, k \in \{1, \ldots, 9\}$:
    \[ \sum_{i=1}^{9} v_{i,j,k} = 1 \]

• Each value used exactly once in each $3 \times 3$ sub-grid:
  – For $i, j \in \{0, 1, 2\}, k \in \{1, \ldots, 9\}$:
    \[ \sum_{r=1}^{3} \sum_{s=1}^{3} v_{3i+r,3j+s,k} = 1 \]

• Q: how to (propositionally) encode Equals1 constraints?
Constraints for fixed cells

```
5 3 7
6 1 9 5
9 8 6
8 6 3
4 8 3 1
7 2 6
6 2 8
4 1 9 5
8 7 9
```
Constraints for fixed cells

- Integer variables:

  \[ v_{1,1} = 5, \quad v_{1,2} = 3, \quad v_{1,5} = 7, \quad v_{2,1} = 6, \quad v_{2,4} = 1, \quad v_{2,5} = 9 \]
  \[ v_{2,6} = 5, \quad v_{3,2} = 9, \quad v_{3,3} = 8, \quad v_{3,8} = 6, \quad v_{4,1} = 8, \quad v_{4,5} = 6, \ldots \]
Constraints for fixed cells

- Integer variables:
  \[ v_{1,1} = 5, \quad v_{1,2} = 3, \quad v_{1,5} = 7, \quad v_{2,1} = 6, \quad v_{2,4} = 1, \quad v_{2,5} = 9 \]
  \[ v_{2,6} = 5, \quad v_{3,2} = 9, \quad v_{3,3} = 8, \quad v_{3,8} = 6, \quad v_{4,1} = 8, \quad v_{4,5} = 6, \ldots \]

- Propositional variables:
  \[ v_{1,1,5} = 1, \quad v_{1,2,3} = 1, \quad v_{1,5,7} = 1, \quad v_{2,1,6} = 1, \quad v_{2,4,1} = 1, \quad v_{2,5,9} = 1 \]
  \[ v_{2,6,5} = 1, \quad v_{3,2,9} = 1, \quad v_{3,3,8} = 1, \quad v_{3,8,6} = 1, \quad v_{4,1,8} = 1, \quad v_{4,5,6} = 1, \ldots \]
Demo
Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples
How to translate to CNF?

- **Obs:** There are no CNF formulas \[\text{Stu13}\]
- **Standard textbook solution**
  - Operator elimination; De Morgan's laws, remove double negations & apply distributivity
  - Worst-case exponential
  - Set of variables constant
- **Tseitin's translation & variants (next)**
  - New variables added
  - Satisfiability is preserved
  - Linear size transformation
How to translate to CNF?

- **Obs:** There are no CNF formulas [Stu13]
How to translate to CNF?

• **Obs:** *There are no CNF formulas*  

• **Standard textbook solution**
  - Operator elimination; De Morgan’s laws, remove double negations & apply distributivity
  - Worst-case exponential
  - Set of variables constant
How to translate to CNF?

• **Obs:** *There are no CNF formulas* [Stu13]

• **Standard textbook solution**
  - Operator elimination; De Morgan’s laws, remove double negations & apply distributivity
  - Worst-case exponential
  - Set of variables constant

• **Tseitin’s translation & variants** (next)
  - New variables added
  - Satisfiability is preserved
  - Linear size transformation
Representing Boolean formulas / circuits I

- Satisfiability problems can be defined on Boolean circuits/formulas
  - Can use any logic connective: \(\land, \lor, \neg, \to, \leftrightarrow, \ldots\)
- Can represent circuits/formulas as CNF formulas
  - For each (simple) gate, CNF formula encodes the consistent assignments to the gate’s inputs and output
    - Given \(z = \text{OP}(x, y)\), represent in CNF \(z \leftrightarrow \text{OP}(x, y)\)
    - CNF formula for the circuit is the conjunction of CNF formula for each gate

\[
F_c = (a \lor c) \land (b \lor c) \land (\bar{a} \lor \bar{b} \lor \bar{c})
\]

\[
F_t = (\bar{r} \lor t) \land (\bar{s} \lor t) \land (r \lor s \lor \bar{t})
\]
Representing Boolean formulas / circuits II

\[ F_c = (a \vee c) \land (b \vee c) \land (\bar{a} \vee \bar{b} \vee \bar{c}) \]
• CNF formula for the circuit is the conjunction of the CNF formula for each gate
  – Can specify objectives with additional clauses

\[
F = (a \lor x) \land (b \lor x) \land (\bar{a} \lor \bar{b} \lor \bar{x}) \land \\
(x \lor \bar{y}) \land (c \lor \bar{y}) \land (\bar{x} \lor \bar{c} \lor y) \land \\
(\bar{y} \lor z) \land (\bar{d} \lor z) \land (y \lor d \lor \bar{z}) \land (z)
\]
Representing Boolean formulas / circuits III

- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses

\[
F = (a \lor x) \land (b \lor x) \land (\bar{a} \lor \bar{b} \lor \bar{x}) \land \\
(x \lor \bar{y}) \land (c \lor \bar{y}) \land (\bar{x} \lor \bar{c} \lor y) \land \\
(\bar{y} \lor z) \land (\bar{d} \lor z) \land (y \lor d \lor \bar{z}) \land (z)
\]

- Note: \( z = d \lor (c \land (\neg(a \land b))) \)
  - No distinction between Boolean circuits and (non-clausal) formulas, besides adding new variables
Representing Boolean formulas / circuits III

- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses

\[ F = (a \lor x) \land (b \lor x) \land (\bar{a} \lor \bar{b} \lor \bar{x}) \land (x \lor \bar{y}) \land (c \lor \bar{y}) \land (\bar{x} \lor \bar{c} \lor y) \land (\bar{y} \lor z) \land (d \lor z) \land (y \lor d \lor \bar{z}) \land (z) \]

- Note: \( z = d \lor (c \land (\neg (a \land b))) \)
  - **No** distinction between Boolean circuits and (non-clausal) formulas, besides adding new variables
- Easy to do more structures: ITEs; Adders; etc.
Quiz – how to encode a 100 input gate?

\[
\begin{align*}
&\text{• Impractical to create the truth table...} \\
&\text{• For any } x_i, \text{ if } x_i = 0, \text{ then } z = 0, \text{ i.e. } \neg x_i \rightarrow \neg z \\
&\text{• If for all } i, x_i = 1, \text{ then } z = 1, \text{ i.e. } \land \Big( x_1 \lor \cdots \lor x_{100} \lor z \Big) \\
&\text{• Resulting CNF encoding:} \\
&\quad 100 \land \Big( x_1 \lor z \Big) \land \Big( x_{100} \lor \cdots \lor x_{100} \lor z \Big) \\
&\text{• Similar ideas apply for other (simple) logical operators: AND, NAND, OR, NOR, etc.}
\end{align*}
\]
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...

Resulting CNF encoding:

\[
100 \land \bigwedge_{i=1}^{100} (x_i \lor z) \land \bigwedge (x_1 \lor \cdots \lor x_{100} \lor z)
\]

Similar ideas apply for other (simple) logical operators: AND, NAND, OR, NOR, etc.
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any $x_i$, if $x_i = 0$, then $z = 0$
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any $x_i$, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any $x_i$, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all $i$ $x_i = 1$, then $z = 1$
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any $x_i$, if $x_i = 0$, then $z = 0$ , i.e. $\neg x_i \rightarrow \neg z$
- If for all $i$ $x_i = 1$, then $z = 1$ , i.e. $\land_i x_i \rightarrow z$
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any \( x_i \), if \( x_i = 0 \), then \( z = 0 \), i.e. \( \neg x_i \rightarrow \neg z \)
- If for all \( i \) \( x_i = 1 \), then \( z = 1 \), i.e. \( \land i x_i \rightarrow z \)
- Resulting CNF encoding:

\[
\bigwedge_{i=1}^{100} (x_i \lor \bar{z}) \land (\bar{x}_1 \lor \cdots \lor \bar{x}_{100} \lor z)
\]
Quiz – how to encode a 100 input gate?

- Impractical to create the truth table...
- For any $x_i$, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all $i$ $x_i = 1$, then $z = 1$, i.e. $\land_i x_i \rightarrow z$
- Resulting CNF encoding:

$$
\bigwedge_{i=1}^{100} (x_i \lor \overline{z}) \land (\overline{x_1} \lor \cdots \lor \overline{x_{100}} \lor z)
$$

- Similar ideas apply for other (simple) logical operators: AND, NAND, OR, NOR, etc.
Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples
Hard vs. soft constraints

- **Hard**: Constraints that *must* be satisfied

An example:

\[
\min \sum_{j=1}^{n} c_j x_j
\]

s.t.

- **Hard constraints**: \( \phi \)
- **Soft constraints**: \((x_j), \text{each with cost } c_j\)
Hard vs. soft constraints

- **Hard**: Constraints that *must* be satisfied
- **Soft**: Constraints that *we would like to satisfy, if possible*
  - Associate a *cost* (can be *unit*) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as $\infty$
Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as $\infty$

- An example:
  - How to model linear cost function optimization?

\[
\begin{align*}
\text{min} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \varphi
\end{align*}
\]
Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as $\infty$

- An example:
  - How to model linear cost function optimization?

$$\begin{align*}
\text{min} & \quad \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} & \quad \varphi
\end{align*}$$

- **Hard** constraints: $\varphi$
Hard vs. soft constraints

- **Hard**: Constraints that *must* be satisfied
- **Soft**: Constraints that *we would like to satisfy, if possible*
  - Associate a *cost* (can be *unit*) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as $\infty$

- An example:
  - How to model linear cost function optimization?

\[
\begin{align*}
\min & \quad \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} & \quad \varphi
\end{align*}
\]

- **Hard** constraints: $\varphi$
- **Soft** constraints: $(\bar{x}_j)$, each with cost $c_j$
Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples
Linear constraints

- **Cardinality constraints**: $\sum_{j=1}^{n} x_j \leq k$
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$
Linear constraints

- **Cardinality constraints**: $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<,\leq,=,\geq,>\}$

- **Pseudo-Boolean constraints**: $\sum_{j=1}^{n} a_j x_j \bowtie k$, with $\bowtie \in \{<,\leq,=,\geq,>\}$

• If variables are non-Boolean, e.g. with finite domain
  - Need to encode variables (more later)
Linear constraints

- **Cardinality constraints**: \[ \sum_{j=1}^{n} x_j \leq k \]
  - How to handle *AtMost1* constraints, \[ \sum_{j=1}^{n} x_j \leq 1 \]?
  - General form: \[ \sum_{j=1}^{n} x_j \bowtie k \], with \[ \bowtie \in \{<, \leq, =, \geq, >\} \]

- **Pseudo-Boolean constraints**: \[ \sum_{j=1}^{n} a_j x_j \bowtie k \]
  - \[ \bowtie \in \{<, \leq, =, \geq, >\} \]

- If variables are non-Boolean, e.g. with finite domain
  - Need to encode variables
    (more later)
Equals1, AtLeast1 & AtMost1 constraints

- \( \sum_{j=1}^{n} x_j = 1 \): encode with \((\sum_{j=1}^{n} x_j \leq 1) \land (\sum_{j=1}^{n} x_j \geq 1)\)

- \( \sum_{j=1}^{n} x_j \geq 1 \): encode with \((x_1 \lor x_2 \lor \ldots \lor x_n)\)

- \( \sum_{j=1}^{n} x_j \leq 1 \) encode with:
  - Pairwise encoding
    - Clauses: \( O(n^2) \) ; No auxiliary variables
  - Sequential counter [Sin05]
    - Clauses: \( O(n) \) ; Auxiliary variables: \( O(n) \)
  - Bitwise encoding [FP01, Pre07]
    - Clauses: \( O(n \log n) \) ; Auxiliary variables: \( O(\log n) \)
  - ...

Pairwise encoding

- How to (propositionally) encode AtMost1 constraint
  \[ a + b + c + d \leq 1? \]
**Pairwise encoding**

- How to (propositionally) encode AtMost1 constraint $a + b + c + d \leq 1$?

  $$
  a \rightarrow \bar{b} \land \bar{c} \land \bar{d} \quad \implies \quad (\bar{a} \lor \bar{b}) \land (\bar{a} \lor \bar{c}) \land (\bar{a} \lor \bar{d})
  $$

  $$
  b \rightarrow \bar{c} \land \bar{d} \land \bar{a} \quad \implies \quad (\bar{b} \lor \bar{c}) \land (\bar{b} \lor \bar{d}) \land (\bar{b} \lor \bar{a})
  $$

  $$
  c \rightarrow \bar{d} \land \bar{a} \land \bar{b} \quad \implies \quad (\bar{c} \lor \bar{d}) \land (\bar{c} \lor \bar{a}) \land (\bar{c} \lor \bar{b})
  $$

  $$
  d \rightarrow \bar{a} \land \bar{b} \land \bar{c} \quad \implies \quad (\bar{d} \lor \bar{a}) \land (\bar{d} \lor \bar{b}) \land (\bar{d} \lor \bar{c})
  $$

- Encoded as: $(\bar{a} \lor \bar{b}) \land (\bar{a} \lor \bar{c}) \land (\bar{a} \lor \bar{d}) \land (\bar{b} \lor \bar{c}) \land (\bar{b} \lor \bar{d}) \land (\bar{c} \lor \bar{d})$
Pairwise encoding

- How to (propositionally) encode AtMost1 constraint \( a + b + c + d \leq 1 \)?

\[
\begin{align*}
  a & \rightarrow \bar{b} \land \bar{c} \land \bar{d} \quad \Rightarrow \quad (\bar{a} \lor \bar{b}) \land (\bar{a} \lor \bar{c}) \land (\bar{a} \lor \bar{d}) \\
  b & \rightarrow \bar{c} \land \bar{d} \land \bar{a} \quad \Rightarrow \quad (\bar{b} \lor \bar{c}) \land (\bar{b} \lor \bar{d}) \land (\bar{b} \lor \bar{a}) \\
  c & \rightarrow \bar{d} \land \bar{a} \land \bar{b} \quad \Rightarrow \quad (\bar{c} \lor \bar{d}) \land (\bar{c} \lor \bar{a}) \land (\bar{c} \lor \bar{b}) \\
  d & \rightarrow \bar{a} \land \bar{b} \land \bar{c} \quad \Rightarrow \quad (\bar{d} \lor \bar{a}) \land (\bar{d} \lor \bar{b}) \land (\bar{d} \lor \bar{c})
\end{align*}
\]

- Encoded as: \( (\bar{a} \lor \bar{b}) \land (\bar{a} \lor \bar{c}) \land (\bar{a} \lor \bar{d}) \land (\bar{b} \lor \bar{c}) \land (\bar{b} \lor \bar{d}) \land (\bar{c} \lor \bar{d}) \)

- With \( N \) variables, number of clauses becomes \( \frac{n(n-1)}{2} \)
  - But no additional variables
Sequential counter encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter:

$$\quad (\bar{x}_1 \lor s_1) \land (\bar{x}_n \lor \bar{s}_{n-1}) \land$$
$$\land_{1<i<n} ((\bar{x}_i \lor s_i) \land (\bar{s}_{i-1} \lor s_i) \land (\bar{x}_i \lor \bar{s}_{i-1}))$$

- If some $x_j = 1$, then all $s_i$ variables must be assigned
  - $s_i = 1$ for $i \geq j$, and so $x_i = 0$ for $i > j$
  - $s_i = 0$ for $i < j$, and so $x_i = 0$ for $i < j$
  - Thus, all other $x_i$ variables must take value 0
- If all $x_j = 0$, can find consistent assignment to $s_i$ variables
- $O(n)$ clauses; $O(n)$ auxiliary variables
Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:

- Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)

- If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j-1$ $x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}) \iff (\overline{x}_j \lor (v_i \leftrightarrow b_i)) = (\overline{x}_j \lor l_i), i = 0, \ldots, r-1$, where $\overline{l_i} \equiv v_i$, if $b_i = 1$ $\overline{l_i} \equiv \overline{v_i}$, otherwise

- If $x_j = 1$, assignment to $v_i$ variables must encode $j-1$ $\for$ consistency, all other $x$ variables must not take value 1

- If all $x_j = 0$, any assignment to $v_i$ variables is consistent

- $O(n \log n)$ clauses; $O(\log n)$ auxiliary variables

- An example: $x_1 + x_2 + x_3 \leq 1$
Bitwise encoding

• Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$ ; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$
    $x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}) \iff (\bar{x}_j \lor (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}))$

• An example: $x_1 + x_2 + x_3 \leq 1$

<table>
<thead>
<tr>
<th>$j - 1$</th>
<th>$v_1 v_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0 00</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1 01</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2 10</td>
</tr>
</tbody>
</table>
Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$ ; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$
    $x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}) \iff (\bar{x}_j \lor (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}))$
  - Clauses $(\bar{x}_j \lor (v_i \leftrightarrow b_i)) = (\bar{x}_j \lor l_i)$, $i = 0, \ldots, r - 1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise

- An example: $x_1 + x_2 + x_3 \leq 1$

<table>
<thead>
<tr>
<th>$j - 1$</th>
<th>$v_1 v_0$</th>
<th>( (\bar{x}_1 \lor \bar{v}_1) \land (\bar{x}_1 \lor \bar{v}_0) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1</td>
<td>01</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$ ; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$
    
    $x_j \rightarrow (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}) \iff (\bar{x}_j \lor (v_0 = b_0) \land \ldots \land (v_{r-1} = b_{r-1}))$
  
  - Clauses $(\bar{x}_j \lor (v_i \leftrightarrow b_i)) = (\bar{x}_j \lor l_i)$, $i = 0, \ldots, r - 1$, where
    
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise
  
  - If $x_j = 1$, assignment to $v_i$ variables must encode $j - 1$
    
    - For consistency, all other $x$ variables must not take value 1
  
  - If all $x_j = 0$, any assignment to $v_i$ variables is consistent
  
  - $O(n \log n)$ clauses ; $O(\log n)$ auxiliary variables

- An example: $x_1 + x_2 + x_3 \leq 1$

<table>
<thead>
<tr>
<th>$j - 1$</th>
<th>$v_1 v_0$</th>
<th>$x_1 \lor v_1 \lor (\bar{x}_1 \lor \bar{v}_1)$</th>
<th>$x_2 \lor v_1 \lor (\bar{x}_2 \lor \bar{v}_1)$</th>
<th>$x_3 \lor v_1 \lor (\bar{x}_3 \lor \bar{v}_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>0 00</td>
<td>$(\bar{x}_1 \lor \bar{v}_1) \land (\bar{x}_1 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_2 \lor \bar{v}_1) \land (\bar{x}_2 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_3 \lor \bar{v}_1) \land (\bar{x}_3 \lor \bar{v}_0)$</td>
</tr>
<tr>
<td>$x_2$</td>
<td>1 01</td>
<td>$(\bar{x}_1 \lor \bar{v}_1) \land (\bar{x}_1 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_2 \lor \bar{v}_1) \land (\bar{x}_2 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_3 \lor \bar{v}_1) \land (\bar{x}_3 \lor \bar{v}_0)$</td>
</tr>
<tr>
<td>$x_3$</td>
<td>2 10</td>
<td>$(\bar{x}_1 \lor \bar{v}_1) \land (\bar{x}_1 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_2 \lor \bar{v}_1) \land (\bar{x}_2 \lor \bar{v}_0)$</td>
<td>$(\bar{x}_3 \lor \bar{v}_1) \land (\bar{x}_3 \lor \bar{v}_0)$</td>
</tr>
</tbody>
</table>
General cardinality constraints

- General form: $\sum_{j=1}^{n} x_j \leq k$ (or $\sum_{j=1}^{n} x_j \geq k$)
  - Operational encoding
    - Clauses/Variables: $O(n)$
    - Does not ensure arc-consistency
  - Generalized pairwise
    - Clauses: $O(2^n)$; no auxiliary variables
  - Sequential counters
    - Clauses/Variables: $O(nk)$
  - BDDs
    - Clauses/Variables: $O(nk)$
  - Sorting networks
    - Clauses/Variables: $O(n \log^2 n)$
  - Cardinality Networks:
    - Clauses/Variables: $O(n \log^2 k)$
  - Pairwise Cardinality Networks:
  - ...
Generalized pairwise encoding

- General form: \( \sum_{j=1}^{n} x_j \leq k \)
- Any combination of \( k + 1 \) true variables is disallowed
Generalized pairwise encoding

- General form: \( \sum_{j=1}^{n} x_j \leq k \)
- Any combination of \( k + 1 \) true variables is disallowed
- Example: \( a + b + c + d \leq 2 \)
Generalized pairwise encoding

- General form: $\sum_{j=1}^{n} x_j \leq k$
- Any combination of $k + 1$ true variables is disallowed
- Example: $a + b + c + d \leq 2$

\[
\begin{align*}
    a \land b & \rightarrow \bar{c} \quad \implies \quad (\bar{a} \lor \bar{b} \lor \bar{c}) \\
    a \land b & \rightarrow \bar{d} \quad \implies \quad (\bar{a} \lor \bar{b} \lor \bar{d}) \\
    a \land c & \rightarrow \bar{d} \quad \implies \quad (\bar{a} \lor \bar{c} \lor \bar{d}) \\
    b \land c & \rightarrow \bar{d} \quad \implies \quad (\bar{b} \lor \bar{c} \lor \bar{d})
\end{align*}
\]

- Encoded as: $(\bar{a} \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor \bar{d}) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor \bar{d})$
Generalized pairwise encoding

- General form: \( \sum_{j=1}^{n} x_j \leq k \)
- Any combination of \( k + 1 \) true variables is disallowed

- Example: \( a + b + c + d \leq 2 \)

\[
\begin{align*}
  a \land b \rightarrow \bar{c} & \implies (\bar{a} \lor \bar{b} \lor \bar{c}) \\
  a \land b \rightarrow \bar{d} & \implies (\bar{a} \lor \bar{b} \lor \bar{d}) \\
  a \land c \rightarrow \bar{d} & \implies (\bar{a} \lor \bar{c} \lor \bar{d}) \\
  b \land c \rightarrow \bar{d} & \implies (\bar{b} \lor \bar{c} \lor \bar{d})
\end{align*}
\]

- Encoded as: \( (\bar{a} \lor \bar{b} \lor \bar{c}) \land (\bar{a} \lor \bar{b} \lor \bar{d}) \land (\bar{a} \lor \bar{c} \lor \bar{d}) \land (\bar{b} \lor \bar{c} \lor \bar{d}) \)

- In general, number of clauses is \( C_{k+1}^n \)
  - Recall: for AtMost1 (i.e. for \( k = 1 \)), number of clauses is: \( \frac{n(n-1)}{2} \)
Another example

• Example: $a + b + c + d + e \leq 2$

• Encoding will contain $C_3^5 = 10$ clauses:

\[
\begin{align*}
  a \land b \rightarrow \bar{c} & \implies (\bar{a} \lor \bar{b} \lor \bar{c}) \\
  a \land b \rightarrow \bar{d} & \implies (\bar{a} \lor \bar{b} \lor \bar{d}) \\
  a \land b \rightarrow \bar{e} & \implies (\bar{a} \lor \bar{b} \lor \bar{e}) \\
  a \land c \rightarrow \bar{d} & \implies (\bar{a} \lor \bar{c} \lor \bar{d}) \\
  a \land c \rightarrow \bar{e} & \implies (\bar{a} \lor \bar{c} \lor \bar{e}) \\
  a \land d \rightarrow \bar{e} & \implies (\bar{a} \lor \bar{d} \lor \bar{e}) \\
  b \land c \rightarrow \bar{d} & \implies (\bar{b} \lor \bar{c} \lor \bar{d}) \\
  b \land c \rightarrow \bar{e} & \implies (\bar{b} \lor \bar{c} \lor \bar{e}) \\
  b \land d \rightarrow \bar{e} & \implies (\bar{b} \lor \bar{d} \lor \bar{e}) \\
  c \land d \rightarrow \bar{e} & \implies (\bar{c} \lor \bar{d} \lor \bar{e})
\end{align*}
\]
• Encode $\sum_{j=1}^{n} x_j \leq k$ with sequential counter:

$$s_i = \sum_{j=1}^{i} x_j$$

$s_i$ represented in unary

$S_{i,1} = S_{i-1,1} \lor x_i$

$S_{i,j} = S_{i-1,j} \lor S_{i-1,j-1} \land x_i$

$v_i = (s_{i-1,k} \land x_i) = 0$
Sequential counter – revisited II

• CNF formula for $\sum_{j=1}^{n} x_j \leq k$:
  
  - Assume: $k > 0 \land n > 1$
  - Indeces: $1 < i < n$, $1 < j \leq k$

  $$(\neg x_1 \lor x_{1,1})$$
  $$(\neg s_{1,j})$$
  $$(\neg x_i \lor s_{i,1})$$
  $$(\neg s_{i-1,1} \lor s_{i,1})$$
  $$(\neg x_i \lor \neg s_{i-1,j-1} \lor s_{i,j})$$
  $$(\neg s_{i-1,j} \lor s_{i,j})$$
  $$(\neg x_i \lor \neg s_{i-1,k})$$
  $$(\neg x_n \lor \neg s_{n-1,k})$$

• $O(n \cdot k)$ clauses & variables
Pseudo-Boolean constraints

- General form: \( \sum_{j=1}^{n} a_j x_j \leq b \)
  - Operational encoding
    - Clauses/Variables: \( O(n) \)
    - Does not guarantee arc-consistency
  - BDDs
    - Worst-case exponential number of clauses

[War98]
[ES06]

\( \nu(n) = \log(n) \log(a_{\text{max}}) \)

Clauses: \( O(n^3 \nu(n)) \)
Aux variables: \( O(n^2 \nu(n)) \)

Improved polynomial watchdog encoding

[ANO+12]

Clauses & aux variables: \( O(n^3 \log(a_{\text{max}})) \)
Pseudo-Boolean constraints

- General form: \( \sum_{j=1}^{n} a_j x_j \leq b \)
  - Operational encoding
    - Clauses/Variables: \( O(n) \)
    - Does not guarantee arc-consistency
  - BDDs
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding
    - Let \( \nu(n) = \log(n) \log(a_{\text{max}}) \)
    - Clauses: \( O(n^3 \nu(n)) \); Aux variables: \( O(n^2 \nu(n)) \)

- Improved polynomial watchdog encoding
  - Clauses & aux variables: \( O(n^3 \log(a_{\text{max}})) \)

[War98] [ES06] [BBR09]
Pseudo-Boolean constraints

• General form: \( \sum_{j=1}^{n} a_j x_j \leq b \)
  - Operational encoding
    ▶ Clauses/Variables: \( \mathcal{O}(n) \)
    ▶ Does not guarantee arc-consistency
  - BDDs
    ▶ Worst-case exponential number of clauses
  - Polynomial watchdog encoding
    ▶ Let \( \nu(n) = \log(n) \log(a_{\text{max}}) \)
    ▶ Clauses: \( \mathcal{O}(n^3 \nu(n)) \); Aux variables: \( \mathcal{O}(n^2 \nu(n)) \)
  - Improved polynomial watchdog encoding
    ▶ Clauses & aux variables: \( \mathcal{O}(n^3 \log(a_{\text{max}})) \)
  - ...

[War98]

[ES06]

[BBR09]

[ANO\textsuperscript{+} 12]
Encoding PB constraints with BDDs

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD
Encoding PB constraints with BDDs

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD
• Encode $3x_1 + 3x_2 + x_3 \leq 3$
• Extract ITE-based circuit from BDD
• Simplify and create final circuit:
More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?

  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...

  - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint

  - Cannot find all consequences in polynomial time [FS02, Tri03, Sel03]

  - Otherwise $P = NP$

- Example: $4x_1 + 3x_2 + 2x_3 = 5$

  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)$

  - Let $x_2 = 0$

  - Either constraint can still be satisfied, but not both
• How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    ▶ $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time
      (Otherwise $P = NP$)
      [FS02, Tri03, Sel03]
More on PB constraints

● How about \( \sum_{j=1}^{n} a_j x_j = k \) ?
  
  – Can use \((\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)\), but...
    
    ▶ \( \sum_{j=1}^{n} a_j x_j = k \) is a knapsack constraint
    
    ▶ Cannot find all consequences in polynomial time
      
      (Otherwise \( P = NP \))

● Example:

\[
4x_1 + 3x_2 + 2x_3 = 5
\]
More on PB constraints

• How about $\sum_{j=1}^{n} a_j x_j = k$ ?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    ▶ $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    ▶ Cannot find all consequences in polynomial time [FS02, Tri03, Sel03]
      (Otherwise $P = NP$)

• Example:

  \[ 4x_1 + 3x_2 + 2x_3 = 5 \]
  
  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)$
More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time  
      (Otherwise P = NP)

- Example:
  
  $4x_1 + 3x_2 + 2x_3 = 5$
  
  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)$
  - Let $x_2 = 0$
More on PB constraints

• How about \( \sum_{j=1}^{n} a_j x_j = k \)?
  - Can use \((\sum_{j=1}^{n} a_j x_j \geq k) \land (\sum_{j=1}^{n} a_j x_j \leq k)\), but...
    - \(\sum_{j=1}^{n} a_j x_j = k\) is a knapsack constraint
    - Cannot find all consequences in polynomial time
      (Otherwise P = NP)

• Example:

  \[4x_1 + 3x_2 + 2x_3 = 5\]

  - Replace by \((4x_1 + 3x_2 + 2x_3 \geq 5) \land (4x_1 + 3x_2 + 2x_3 \leq 5)\)
  - Let \(x_2 = 0\)
  - Either constraint can still be satisfied, but not both
Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples
CSP constraints

- Many possible encodings:
  - Direct encoding
    - References: [dK89, GJ96, Wal00]
  - Log encoding
    - Reference: [Wal00]
  - Support encoding
    - References: [Kas90, Gen02]
  - Log-Support encoding
    - Reference: [Gav07]
  - Order encoding for finite linear CSPs
    - Reference: [TTKB09]
Direct encoding for CSP w/ binary constraints

- Variable \( x_i \) with domain \( D_i \), with \( m_i = |D_i| \)

- Constraints are relations over domains of variables
  - For a constraint over \( x_1, \ldots, x_k \), define relation \( R \subseteq D_1 \times \cdots \times D_k \)
  - Need to encode elements not in the relation
  - For a binary relation, use set of binary clauses, one for each element not in \( R \)

- Represent values of \( x_i \) with Boolean variables \( x_{i,1}, \ldots, x_{i,m_i} \)

- Require \( \sum_{k=1}^{m_i} x_{i,k} = 1 \)
  - Suffices to require \( \sum_{k=1}^{m_i} x_{i,k} \geq 1 \)  

- If the pair of assignments \( x_i = v_i \land x_j = v_j \) is not allowed, add binary clause \((\overline{x_i,v_i} \lor \overline{x_j,v_j})\)
• Encoding problems to SAT is ubiquitous:
  – Many more encodings of finite domain CSP into SAT
  – Encodings of Answer Set Programming (ASP) into SAT
  – Eager SMT solving
  – Theorem provers iteratively encode problems into SAT
  – Model finders iteratively encode problems into SAT
  – ...

Minimum vertex cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover $U$ of minimum size
Minimum vertex cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover $U$ of minimum size

Vertex cover: $\{v_2, v_3, v_4\}$
Minimum vertex cover

- The problem:
  - Graph \( G = (V, E) \)
  - Vertex cover \( U \subseteq V \)
    - For each \((v_i, v_j) \in E\), either \(v_i \in U\) or \(v_j \in U\)
  - Minimum vertex cover: vertex cover \( U \) of minimum size

Vertex cover: \( \{v_2, v_3, v_4\} \)
Min vertex cover: \( \{v_1\} \)
Minimum vertex cover

- Modeling with **Pseudo-Boolean Optimization (PBO)**:
  - Variables: \( x_i \) for each \( v_i \in V \), with \( x_i = 1 \) iff \( v_i \in U \)
  - Clauses: \( (x_i \lor x_j) \) for each \( (v_i, v_j) \in E \)
  - Objective function: minimize number of **true** \( x_i \) variables
    - I.e. minimize vertices included in \( U \)
Minimum vertex cover

- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: $x_i$ for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Clauses: $(x_i \lor x_j)$ for each $(v_i, v_j) \in E$
  - Objective function: minimize number of true $x_i$ variables
    - i.e. minimize vertices included in $U$

![Graph](image)

minimize $x_1 + x_2 + x_3 + x_4$

subject to $(x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4)$
Minimum vertex cover

- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: \( x_i \) for each \( v_i \in V \), with \( x_i = 1 \) iff \( v_i \in U \)
  - Clauses: \( (x_i \lor x_j) \) for each \( (v_i, v_j) \in E \)
  - Objective function: minimize number of true \( x_i \) variables
    - I.e. minimize vertices included in \( U \)

\[
\begin{align*}
\text{minimize} & \quad x_1 + x_2 + x_3 + x_4 \\
\text{subject to} & \quad (x_1 \lor x_2) \land (x_1 \lor x_3) \land (x_1 \lor x_4)
\end{align*}
\]

- Alternative propositional encoding:
  \[
  \begin{align*}
  \varphi_S &= \{\neg x_1, \neg x_2, \neg x_3, \neg x_4\} \\
  \varphi_H &= \{(x_1 \lor x_2), (x_1 \lor x_3), (x_1 \lor x_4)\}
  \end{align*}
\]
Graph coloring

• Given undirected graph $G = (V, E)$ and $k$ colors:
  – Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

• How to model color assignments to vertices?
  – $x_{ij} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

• How to model adjacent vertices with different colors?
  – $(\neg x_{ij} \lor \neg x_{lj})$ if $(v_i, v_l) \in E$, with $j \in \{1, \ldots, k\}$

• How to model vertices get some color?
  – $\sum_{j \in \{1, \ldots, k\}} x_{ij} = 1$, for $v_i \in V$

Note: it suffices to use $(\bigvee_{j \in \{1, \ldots, k\}} x_{ij})$
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

![Valid coloring](image1.png)

![Invalid coloring](image2.png)
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

- How to model color assignments to vertices?

  - $x_{ij} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

- How to model adjacent vertices with different colors?

  - $(\neg x_{ij} \lor \neg x_{lj})$ if $(v_i, v_l) \in E$, with $j \in \{1, \ldots, k\}$

- How to model vertices get some color?

  - $\sum_{j \in \{1, \ldots, k\}} x_{ij} = 1$, for $v_i \in V$

  - Note: it suffices to use $(\bigvee_{j \in \{1, \ldots, k\}} x_{ij})$
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

  ![Valid coloring](image)

  ![Invalid coloring](image)

- How to model color assignments to vertices?
  - $x_{i,j} = 1 \iff \text{vertex } v_i \in V \text{ is assigned color } j \in \{1, \ldots, k\}$
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

  ![Valid and Invalid colorings](image)

- How to model color assignments to vertices?
  - $x_{i,j} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

- How to model adjacent vertices with different colors?
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

![Valid and Invalid colorings]

- How to model color assignments to vertices?
  - $x_{i,j} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \lor \neg x_{l,j})$ if $(v_i, v_l) \in E$, with $j \in \{1, \ldots, k\}$
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

[Diagram showing valid and invalid colorings]

- How to model color assignments to vertices?
  - $x_{i,j} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \lor \neg x_{l,j})$ if $(v_i, v_l) \in E$, with $j \in \{1, \ldots, k\}$

- How to model vertices get some color?
Graph coloring

- Given undirected graph \( G = (V, E) \) and \( k \) colors:
  - Can we assign colors to vertices of \( G \) s.t. any pair of adjacent vertices are assigned different colors?

\[
\begin{align*}
\text{Valid coloring} & \quad \text{Invalid coloring}
\end{align*}
\]

- How to model color assignments to vertices?
  - \( x_{i,j} = 1 \) iff vertex \( v_i \in V \) is assigned color \( j \in \{1, \ldots, k\} \)

- How to model adjacent vertices with different colors?
  - \( (\neg x_{i,j} \lor \neg x_{l,j}) \) if \((v_i, v_l) \in E\), with \( j \in \{1, \ldots, k\} \)

- How to model vertices get some color?
  - \( \sum_{j \in \{1, \ldots, k\}} x_{i,j} = 1, \text{ for } v_i \in V \)
Graph coloring

- Given undirected graph $G = (V, E)$ and $k$ colors:
  - Can we assign colors to vertices of $G$ s.t. any pair of adjacent vertices are assigned different colors?

- How to model color assignments to vertices?
  - $x_{i,j} = 1$ iff vertex $v_i \in V$ is assigned color $j \in \{1, \ldots, k\}$

- How to model adjacent vertices with different colors?
  - $(-x_{i,j} \lor -x_{l,j})$ if $(v_i, v_l) \in E$, with $j \in \{1, \ldots, k\}$

- How to model vertices get some color?
  - $\sum_{j \in \{1, \ldots, k\}} x_{i,j} = 1$, for $v_i \in V$
  - Note: it suffices to use $\left( \lor_{j \in \{1, \ldots, k\}} x_{i,j} \right)$
The N-Queens problem I

- The N-Queens Problem:
  Place N queens on a $N \times N$ board, such that no two queens attack each other

- Example for a $5 \times 5$ board:
The N-Queens problem II

- $x_{ij}$: 1 if queen placed in position $(i, j)$; 0 otherwise

- Each row must have exactly one queen:

  $$1 \leq i \leq N, \quad \sum_{j=1}^{N} x_{ij} = 1$$

- Each column must have exactly one queen:

  $$1 \leq j \leq N, \quad \sum_{i=1}^{N} x_{ij} = 1$$

- Also, need to define constraints on diagonals...
Each diagonal can have at most one queen:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  i &= 1, \quad 2 \leq j < N, & \sum_{k=0}^{j-1} x_{i+k-j-k} \leq 1 \\
  i &= N, \quad 1 \leq j < N, & \sum_{k=0}^{N-j} x_{i-k+j+k} \leq 1 \\
  j &= 1, \quad 1 \leq i < N, & \sum_{k=0}^{N-i} x_{i+k-j+k} \leq 1 \\
  j &= N, \quad 2 \leq i < N, & \sum_{k=0}^{i-1} x_{i-k-j-k} \leq 1
\end{align*}
\]
Correct circuit

Faulty circuit

Input stimuli: \(\langle r, s \rangle = \langle 0, 1 \rangle\)
Valid output: \(\langle y, z \rangle = \langle 0, 0 \rangle\)

Input stimuli: \(\langle r, s \rangle = \langle 0, 1 \rangle\)
Invalid output: \(\langle y, z \rangle = \langle 0, 0 \rangle\)

- The model:
  - **Hard** clauses: Input and output values
  - **Soft** clauses: CNF representation of circuit

- The problem:
  - Maximize number of satisfied clauses (i.e. circuit gates)
Software package upgrades

- Universe of software packages: \( \{p_1, \ldots, p_n\} \)
- Associate \( x_i \) with \( p_i \): \( x_i = 1 \) iff \( p_i \) is installed
- Constraints associated with package \( p_i \): \((p_i, D_i, C_i)\)
  - \( D_i \): dependencies (required packages) for installing \( p_i \)
  - \( C_i \): conflicts (disallowed packages) for installing \( p_i \)
- Example problem: **Maximum Installability**
  - Maximum number of packages that can be installed
  - Package constraints represent hard clauses
  - Soft clauses: \((x_i)\)

Package constraints:

\[
(p_1, \{p_2 \lor p_3\}, \{p_4\})
(p_2, \{p_3\}, \{p_4\})
(p_3, \{p_2\}, \emptyset)
(p_4, \{p_2, p_3\}, \emptyset)
\]
Software package upgrades

- Universe of software packages: \( \{ p_1, \ldots, p_n \} \)
- Associate \( x_i \) with \( p_i \): \( x_i = 1 \) iff \( p_i \) is installed
- Constraints associated with package \( p_i \): \( (p_i, D_i, C_i) \)
  - \( D_i \): dependencies (required packages) for installing \( p_i \)
  - \( C_i \): conflicts (disallowed packages) for installing \( p_i \)
- Example problem: Maximum Installability
  - Maximum number of packages that can be installed
  - Package constraints represent hard clauses
  - Soft clauses: \( (x_i) \)

Package constraints:

\[
(p_1, \{ p_2 \lor p_3 \}, \{ p_4 \}) \\
(p_2, \{ p_3 \}, \{ p_4 \}) \\
(p_3, \{ p_2 \}, \emptyset) \\
(p_4, \{ p_2, p_3 \}, \emptyset)
\]

MaxSAT formulation:

\[
\varphi_H = \{(\neg x_1 \lor x_2 \lor x_3), (\neg x_1 \lor \neg x_4), (\neg x_2 \lor x_3), (\neg x_2 \lor \neg x_4), (\neg x_3 \lor x_2), (\neg x_4 \lor x_2), (\neg x_4 \lor x_3)\}
\]

\[
\varphi_S = \{(x_1), (x_2), (x_3), (x_4)\}
\]
The knapsack problem

- Given list of pairs \((v_i, w_i), \ i = 1, \ldots, n\)
  - Each pair \((v_i, w_i)\), represents the value and weight of object \(i\)
The knapsack problem

- Given list of pairs \((v_i, w_i), \ i = 1, \ldots, n\)
  - Each pair \((v_i, w_i)\), represents the value and weight of object \(i\)
- Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed \(W\)
The knapsack problem

- Given list of pairs \((v_i, w_i), \ i = 1, \ldots, n\)
  - Each pair \((v_i, w_i)\), represents the value and weight of object \(i\)
- Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed \(W\)

- Propositional encoding for the knapsack problem?
The knapsack problem

• Given list of pairs \((v_i, w_i), \ i = 1, \ldots, n\)
  
  - Each pair \((v_i, w_i)\), represents the value and weight of object \(i\)

• Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed \(W\)

• Propositional encoding for the knapsack problem?

• **Solution:** consider 0-1 ILP (or PBO) formulation:
  
  - Associate propositional variable \(x_i\) with each objet \(i\)
  - \(x_i = 1\) iff object \(i\) is picked

\[
\begin{align*}
\max & \quad \sum_{i=1}^{n} v_i \cdot x_i \\
\text{s.t} & \quad \sum_{i=1}^{n} w_i \cdot x_i \leq W
\end{align*}
\]
Part 3

Problem Solving with SAT Oracles
Computing a model

- **Q:** How to solve the **FSAT** problem?

**FSAT:** Compute a model of a satisfiable CNF formula \( \mathcal{F} \), using an NP oracle

• Algorithm needs \( |\text{var}(\mathcal{F})| \) calls to an NP oracle

- Note: Cannot solve FSAT with logarithmic number of NP oracle calls, unless \( P = NP \) [GF93]

• FSAT is an example of a function problem
Computing a model

- **Q:** How to solve the FSAT problem?

  **FSAT:** Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle
  
  - A possible algorithm:
    - Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$
    - Consider $\mathcal{F} \land (x_i)$. Call NP oracle. If answer is **yes**, then add $(x_i)$ to $\mathcal{F}$. If answer is **no**, then add $(\neg x_i)$ to $\mathcal{F}$

- **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $P = NP$ [GF93]

- **FSAT** is an example of a function problem

- **Note:** FSAT can be solved with one SAT oracle call
Computing a model

- **Q:** How to solve the **FSAT** problem?

  **FSAT:** Compute a model of a satisfiable CNF formula \( \mathcal{F} \), using an NP oracle
  
  - A possible algorithm:
    
    ▶ Analyze each variable \( x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F}) \)
    
    ▶ Consider \( \mathcal{F} \land (x_i) \). Call NP oracle. If answer is **yes**, then add \((x_i)\) to \( \mathcal{F} \). If answer is **no**, then add \((-x_i)\) to \( \mathcal{F} \)

  - Algorithm needs \( |\text{var}(\mathcal{F})| \) calls to an NP oracle

\[ \text{Note: Cannot solve FSAT with logarithmic number of NP oracle calls, unless P = NP} \ [\text{GF93}] \]
Computing a model

- **Q:** How to solve the FSAT problem?

  **FSAT:** Compute a model of a satisfiable CNF formula \( \mathcal{F} \), using an NP oracle
  
  - A possible algorithm:
    - Analyze each variable \( x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F}) \)
    - Consider \( \mathcal{F} \land (x_i) \). Call NP oracle. If answer is yes, then add \( (x_i) \) to \( \mathcal{F} \). If answer is no, then add \( (\neg x_i) \) to \( \mathcal{F} \)
  
  - Algorithm needs \(|\text{var}(\mathcal{F})|\) calls to an NP oracle

  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless \( P = NP \) \[GF93\]

- FSAT is an example of a **function** problem
Computing a model

• **Q:** How to solve the **FSAT** problem?
  
  **FSAT:** Compute a model of a satisfiable CNF formula \( \mathcal{F} \), using an NP oracle
  
  – A possible algorithm:
    
    ▶ Analyze each variable \( x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F}) \)
    ▶ Consider \( \mathcal{F} \land (x_i) \). Call NP oracle. If answer is yes, then add \((x_i)\) to \( \mathcal{F} \). If answer is no, then add \( (\neg x_i) \) to \( \mathcal{F} \)

  – Algorithm needs \( |\text{var}(\mathcal{F})| \) calls to an NP oracle

  – **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless \( P = \text{NP} \) [GF93]

• **FSAT** is an example of a function problem
  
  – **Note:** FSAT can be solved with one SAT oracle call
### Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
</table>

- **Decision Problems**: Some solution
- **Function Problems**: All solutions
- **Enumeration Problems**: # solutions
- **Counting Problems**:
### Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
</tbody>
</table>
### Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
</tbody>
</table>

Some solution
## Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td><strong>Some solution</strong></td>
<td>Function Problems</td>
</tr>
</tbody>
</table>
## Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td></td>
</tr>
</tbody>
</table>
Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td>Enumeration Problems</td>
</tr>
</tbody>
</table>
## Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td>Enumeration Problems</td>
</tr>
<tr>
<td># solutions</td>
<td></td>
</tr>
</tbody>
</table>
Beyond decision problems

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td>Enumeration Problems</td>
</tr>
<tr>
<td># solutions</td>
<td>Counting Problems</td>
</tr>
</tbody>
</table>
... and beyond NP – decision and function problems

\[
\begin{align*}
\Sigma_3^p & \quad \Delta_3^p & \quad \Pi_3^p \\
\Sigma_2^p & \quad \Delta_2^p & \quad \Pi_2^p \\
\Sigma_1^p & \quad \Delta_1^p & \quad \Pi_1^p = \text{coNP} \\
\Sigma_0^p & \quad \Delta_0^p = \Sigma_0^p = P = \Pi_0^p = \Delta_1^p \\
\Sigma_0^p & \quad \Delta_0^p = \Sigma_0^p = P = \Pi_0^p = \Delta_1^p \\
\end{align*}
\]

\[
\begin{align*}
\text{FNP} & = \text{FΣ}_1^p \\
\text{FP} & = \text{FΠ}_1^p = \text{coFNP} \\
\text{FΔ}_0^p & = \text{FS}_0^p \\
\text{FP} & = \text{FΠ}_0^p = \text{FΔ}_1^p \\
\end{align*}
\]
Oracle-based problem solving – ideal scenario

Poly-time Algorithm

Yes/No + Witness

Bounded # of calls / queries

Decision Procedure

SAT, SMT, CSP, ...
Solver / Oracle
Oracle-based problem solving – in some settings

Poly-time Algorithm

Yes/No + Witness

Decision Procedure

Bounded # of calls / queries

SAT, SMT, CSP, ...
Solver / Oracle
Many problems to solve – within \( \text{FP}^{\text{NP}} \)

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td>Enumeration Problems</td>
</tr>
</tbody>
</table>
Many problems to solve – within \( \text{FP}^{\text{NP}} \)

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes/No</td>
<td>Decision Problems</td>
</tr>
<tr>
<td>Some solution</td>
<td>Function Problems</td>
</tr>
<tr>
<td>All solutions</td>
<td>Enumeration Problems</td>
</tr>
</tbody>
</table>

**Function Problems on Propositional Formulas**

- MaxSAT
- PBO
- \( \ldots \) (ellipsis)
- WBO
- MinSAT
- Autarkies
- Prime Implicates
- Prime Implicates
- Indep. Vars
- Implicates Ext.
- Implicates Ext.
Many problems to solve – within $\text{FP}^{\text{NP}}$

<table>
<thead>
<tr>
<th>Answer</th>
<th>Problem Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yes/No</strong></td>
<td>Decision Problems</td>
</tr>
<tr>
<td><strong>Some solution</strong></td>
<td>Function Problems</td>
</tr>
<tr>
<td><strong>All solutions</strong></td>
<td>Enumeration Problems</td>
</tr>
</tbody>
</table>

### Function Problems on Propositional Formulas

- **Optimization Problems**
  - MaxSAT
  - MinSAT
  - PBO
  - WBO

- **Minimal Sets**
  - Minimal Models
  - Maximal Models
  - Backbones
  - MUSes
  - MCSes
  - MFSes
  - MCFSes
  - Prime Implicates
  - Autarkies
  - Indep. Vars
  - Implicates Ext.
  - Implicant Ext.
  - MDSes
  - MESes
  - MNSes
  - MFSes
  - MCFSes
Selection of topics

- Problem Solving with SAT
  - PBO
  - B&B Search
  - Enumeration
  - OPT SAT
  - Lazy SMT
  - LCG
- Encodings
  - Eager SMT
  - MBD
  - BM
- Oracles
  - MC: ic3
  - CEGAR QBF
  - Counting
  - Enumeration
  - MUS
- Embeddings
  - Min. Models
  - Backbones
- Oracles
  - MC: ic3
  - CEGAR QBF
  - Counting
  - Enumeration
  - MUS
- MUS extraction
- MaxSAT solving

MaxSAT solving
Outline

Minimal Unsatisfiability

Maximum Satisfiability

Examples in PySAT
Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... (hundreds of consistent constraints)</td>
<td></td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td>... (hundreds of consistent constraints)</td>
<td></td>
</tr>
</tbody>
</table>

- Set of constraints consistent / satisfiable?
Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

- Set of constraints consistent / satisfiable? No
## Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

• Set of constraints **consistent / satisfiable? No**
• Minimal subset of constraints that is **inconsistent / unsatisfiable?**
## Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

- Set of constraints **consistent / satisfiable?** No
- Minimal subset of constraints that is **inconsistent / unsatisfiable?**
- Minimal subset of constraints whose removal makes remaining constraints consistent?
### Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

- Set of constraints consistent / satisfiable? **No**
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?
Analyzing inconsistency – timetabling

<table>
<thead>
<tr>
<th>Subject</th>
<th>Day</th>
<th>Time</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intro Prog</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Intro AI</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Databases</td>
<td>Tue</td>
<td>11:00-12:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Linear Alg</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>6.2.46</td>
</tr>
<tr>
<td>Calculus</td>
<td>Tue</td>
<td>10:00-11:00</td>
<td>8.2.37</td>
</tr>
<tr>
<td>Adv Calculus</td>
<td>Mon</td>
<td>9:00-10:00</td>
<td>8.2.06</td>
</tr>
</tbody>
</table>

... (hundreds of consistent constraints)

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?
- How to compute these minimal sets?
Unsatisfiable formulas – MUSes & MCSes

• Given $\mathcal{F} (\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall \mathcal{M}' \subseteq \mathcal{M}, \mathcal{M}' \not\models \bot$

$$\neg x_1 \lor \neg x_2 \land (x_1) \land (x_2) \land (\neg x_3 \lor \neg x_4) \land (x_3) \land (x_4) \land (x_5 \lor x_6)$$
• Given \( F \) (\( \models \bot \)), \( M \subseteq F \) is a **Minimal Unsatisfiable Subset (MUS)** iff \( M \models \bot \) and \( \forall M' \subset M, M' \not\models \bot \)

\[
(\neg x_1 \lor \neg x_2) \land (x_1) \land (x_2) \land (\neg x_3 \lor \neg x_4) \land (x_3) \land (x_4) \land (x_5 \lor x_6)
\]
Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F} (\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall \mathcal{M}' \subseteq \mathcal{M}, \mathcal{M}' \not\models \bot$

$$\neg x_1 \lor \neg x_2 \land (x_1) \land (x_2) \land (\neg x_3 \lor \neg x_4) \land (x_3) \land (x_4) \land (x_5 \lor x_6)$$

- Given $\mathcal{F} (\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall \mathcal{C}' \subseteq \mathcal{C}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $S = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$\neg x_1 \lor \neg x_2 \land (x_1) \land (x_2) \land (\neg x_3 \lor \neg x_4) \land (x_3) \land (x_4) \land (x_5 \lor x_6)$$
Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F} (\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and \( \forall \mathcal{M}' \subset \mathcal{M}, \mathcal{M}' \not\models \bot \)

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

- Given $\mathcal{F} (\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and \( \forall \mathcal{C}' \subset \mathcal{C}, \mathcal{F} \setminus \mathcal{C}' \models \bot \). $S = \mathcal{F} \setminus \mathcal{C}$ is MSS

\[
(\neg x_1 \lor \neg x_2) \land (x_1) \land (x_2) \land (\neg x_3 \lor \neg x_4) \land (x_3) \land (x_4) \land (x_5 \lor x_6)
\]
Unsatisfiable formulas – MUSes & MCSes

• Given $\mathcal{F} (\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall \mathcal{M}' \subset \mathcal{M}, \mathcal{M}' \not\models \bot$

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

• Given $\mathcal{F} (\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall \mathcal{C}' \subset \mathcal{C}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $S = \mathcal{F} \setminus \mathcal{C}$ is MSS

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

• MUSes and MCSes are (subset-)minimal sets

• MUSes and minimal hitting sets of MCSes and vice-versa

[Rei87, BS05]
Unsatisfiable formulas – MUSes & MCSes

- Given $F \models \bot$, $M \subseteq F$ is a Minimal Unsatisfiable Subset (MUS) iff $M \models \bot$ and $\forall M' \subset M, M' \not\models \bot$

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

- Given $F \models \bot$, $C \subseteq F$ is a Minimal Correction Subset (MCS) iff $F \setminus C \not\models \bot$ and $\forall C' \subset C, F \setminus C' \models \bot$. $S = F \setminus C$ is MSS

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

- MUSes and MCSes are (subset-)minimal sets

- MUSes and minimal hitting sets of MCSes and vice-versa

[Rei87, BS05]

- How to compute MUSes & MCSes efficiently with SAT oracles?
Why it matters?

- **Analysis of over-constrained systems**
  - Model-based diagnosis
    - Software fault localization
    - Spreadsheet debugging
    - Debugging relational specifications (e.g. Alloy)
    - Type error debugging
    - Axiom pinpointing in description logics
    - ...
  - Model checking of software & hardware systems
  - Inconsistency measurement
  - Minimal models; MinCost SAT; ...
  - ...

- **Find minimal relaxations to recover consistency**
  - But also minimum relaxations to recover consistency, eg. **MaxSAT**

- **Find minimal explanations of inconsistency**
  - But also minimum explanations of inconsistency, eg. **Smallest MUS**
Deletion-based algorithm

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
\[
\mathcal{M} \leftarrow \mathcal{F}
\]
foreach $c \in \mathcal{M}$ do
\[
\text{if } \neg \text{SAT}(\mathcal{M} \setminus \{c\}) \text{ then }
\]
\[
\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}
\]
// If $\neg \text{SAT}(\mathcal{M} \setminus \{c\})$, then $c \notin \text{MUS}$
return $\mathcal{M}$
// Final $\mathcal{M}$ is MUS
end

- Number of oracles calls: $\mathcal{O}(m)$

[CD91, BDTW93]
Deletion-based algorithm

**Input**: Set $\mathcal{F}$

**Output**: Minimal subset $\mathcal{M}$

begin

\[
\mathcal{M} \leftarrow \mathcal{F}
\]

foreach $c \in \mathcal{M}$ do

\[
\text{if } \neg \text{SAT}(\mathcal{M} \setminus \{c\}) \text{ then}
\]

\[
\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}
\]

// Remove $c$ from $\mathcal{M}$

\[
\text{// Final } \mathcal{M} \text{ is MUS}
\]

end

- Number of oracles calls: $\mathcal{O}(m)$

[CD91, BDTW93]
Deletion – MUS example

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
<th>$c_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg x_1 \lor \neg x_2$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$\neg x_3 \lor \neg x_4$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td>$x_5 \lor x_6$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$\mathcal{M} \setminus {c}$</th>
<th>$\neg \text{SAT(} \mathcal{M} \setminus {c} \rangle$</th>
<th>Outcome</th>
</tr>
</thead>
</table>
Deletion – MUS example

\[
\begin{array}{cccccccc}
  c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
  (\neg x_1 \lor \neg x_2) & (x_1) & (x_2) & (\neg x_3 \lor \neg x_4) & (x_3) & (x_4) & (x_5 \lor x_6) \\
\end{array}
\]

<table>
<thead>
<tr>
<th>(\mathcal{M})</th>
<th>(\mathcal{M}\setminus{c})</th>
<th>(\neg\text{SAT}(\mathcal{M}\setminus{c}))</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c_1..c_7)</td>
<td>(c_2..c_7)</td>
<td>(1)</td>
<td>Drop (c_1)</td>
</tr>
</tbody>
</table>
Deletion – MUS example

\[
\begin{array}{cccccccc}
  c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
  (\neg x_1 \vee \neg x_2) & (x_1) & (x_2) & (\neg x_3 \vee \neg x_4) & (x_3) & (x_4) & (x_5 \vee x_6) \\
\end{array}
\]

<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$\mathcal{M} \setminus {c}$</th>
<th>$\neg \text{SAT}(\mathcal{M} \setminus {c})$</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1..c_7$</td>
<td>$c_2..c_7$</td>
<td>1</td>
<td>Drop $c_1$</td>
</tr>
<tr>
<td>$c_2..c_7$</td>
<td>$c_3..c_7$</td>
<td>1</td>
<td>Drop $c_2$</td>
</tr>
</tbody>
</table>
Deletion – MUS example

\[
\begin{array}{cccccccc}
  c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
  (\neg x_1 \lor \neg x_2) & (x_1) & (x_2) & (\neg x_3 \lor \neg x_4) & (x_3) & (x_4) & (x_5 \lor x_6) \\
\end{array}
\]

<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$\mathcal{M} \setminus {c}$</th>
<th>$\neg \text{SAT}(\mathcal{M} \setminus {c})$</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1..c_7$</td>
<td>$c_2..c_7$</td>
<td>1</td>
<td>Drop $c_1$</td>
</tr>
<tr>
<td>$c_2..c_7$</td>
<td>$c_3..c_7$</td>
<td>1</td>
<td>Drop $c_2$</td>
</tr>
<tr>
<td>$c_3..c_7$</td>
<td>$c_4..c_7$</td>
<td>1</td>
<td>Drop $c_3$</td>
</tr>
</tbody>
</table>
Deletion – MUS example

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

<table>
<thead>
<tr>
<th>( \mathcal{M} )</th>
<th>( \mathcal{M} \setminus {c} )</th>
<th>( \neg \text{SAT}(\mathcal{M} \setminus {c}) )</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1..c_7 )</td>
<td>( c_2..c_7 )</td>
<td>1</td>
<td>Drop ( c_1 )</td>
</tr>
<tr>
<td>( c_2..c_7 )</td>
<td>( c_3..c_7 )</td>
<td>1</td>
<td>Drop ( c_2 )</td>
</tr>
<tr>
<td>( c_3..c_7 )</td>
<td>( c_4..c_7 )</td>
<td>1</td>
<td>Drop ( c_3 )</td>
</tr>
<tr>
<td>( c_4..c_7 )</td>
<td>( c_5..c_7 )</td>
<td>0</td>
<td>Keep ( c_4 )</td>
</tr>
</tbody>
</table>
Deletion – MUS example

\[
\begin{array}{cccccccc}
  c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \\
\hline
  \neg x_1 \lor \neg x_2 & x_1 & x_2 & \neg x_3 \lor \neg x_4 & x_3 & x_4 & x_5 \lor x_6 \\
\end{array}
\]

\[
\begin{array}{cccc}
  M & M\setminus\{c\} & \neg \text{SAT}(M\setminus\{c\}) & \text{Outcome} \\
  \hline
  c_1\ldots c_7 & c_2\ldots c_7 & 1 & \text{Drop } c_1 \\
  c_2\ldots c_7 & c_3\ldots c_7 & 1 & \text{Drop } c_2 \\
  c_3\ldots c_7 & c_4\ldots c_7 & 1 & \text{Drop } c_3 \\
  c_4\ldots c_7 & c_5\ldots c_7 & 0 & \text{Keep } c_4 \\
  c_4\ldots c_7 & c_4 c_6 c_7 & 0 & \text{Keep } c_5 \\
\end{array}
\]
Deletion – MUS example

\[ c_1 \vdash \neg x_1 \lor \neg x_2 \quad c_2 \vdash x_1 \quad c_3 \vdash x_2 \quad c_4 \vdash \neg x_3 \lor \neg x_4 \quad c_5 \vdash x_3 \quad c_6 \vdash x_4 \quad c_7 \vdash x_5 \lor x_6 \]

<table>
<thead>
<tr>
<th>$\mathcal{M}$</th>
<th>$\mathcal{M} \setminus {c}$</th>
<th>$\neg \text{SAT}(\mathcal{M} \setminus {c})$</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1 \ldots c_7$</td>
<td>$c_2 \ldots c_7$</td>
<td>1</td>
<td>Drop $c_1$</td>
</tr>
<tr>
<td>$c_2 \ldots c_7$</td>
<td>$c_3 \ldots c_7$</td>
<td>1</td>
<td>Drop $c_2$</td>
</tr>
<tr>
<td>$c_3 \ldots c_7$</td>
<td>$c_4 \ldots c_7$</td>
<td>1</td>
<td>Drop $c_3$</td>
</tr>
<tr>
<td>$c_4 \ldots c_7$</td>
<td>$c_5 \ldots c_7$</td>
<td>0</td>
<td>Keep $c_4$</td>
</tr>
<tr>
<td>$c_4 \ldots c_7$</td>
<td>$c_4 c_6 c_7$</td>
<td>0</td>
<td>Keep $c_5$</td>
</tr>
<tr>
<td>$c_4 \ldots c_7$</td>
<td>$c_4 c_5 c_7$</td>
<td>0</td>
<td>Keep $c_6$</td>
</tr>
</tbody>
</table>
Deletion – MUS example

\[ \neg x_1 \lor \neg x_2 \times x_1 \times x_2 \times \neg x_3 \lor \neg x_4 \times x_3 \times x_4 \times x_5 \lor x_6 \]

<table>
<thead>
<tr>
<th>( \mathcal{M} )</th>
<th>( \mathcal{M} \setminus {c} )</th>
<th>( \neg \text{SAT}(\mathcal{M} \setminus {c}) )</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1 \ldots c_7 )</td>
<td>( c_2 \ldots c_7 )</td>
<td>1</td>
<td>Drop ( c_1 )</td>
</tr>
<tr>
<td>( c_2 \ldots c_7 )</td>
<td>( c_3 \ldots c_7 )</td>
<td>1</td>
<td>Drop ( c_2 )</td>
</tr>
<tr>
<td>( c_3 \ldots c_7 )</td>
<td>( c_4 \ldots c_7 )</td>
<td>1</td>
<td>Drop ( c_3 )</td>
</tr>
<tr>
<td>( c_4 \ldots c_7 )</td>
<td>( c_5 \ldots c_7 )</td>
<td>0</td>
<td>Keep ( c_4 )</td>
</tr>
<tr>
<td>( c_4 \ldots c_7 )</td>
<td>( c_4 c_6 c_7 )</td>
<td>0</td>
<td>Keep ( c_5 )</td>
</tr>
<tr>
<td>( c_4 \ldots c_7 )</td>
<td>( c_4 c_5 c_7 )</td>
<td>0</td>
<td>Keep ( c_6 )</td>
</tr>
<tr>
<td>( c_4 \ldots c_7 )</td>
<td>( c_4 \ldots c_6 )</td>
<td>1</td>
<td>Drop ( c_7 )</td>
</tr>
</tbody>
</table>
### Deletion – MUS example

<table>
<thead>
<tr>
<th>( \mathcal{M} )</th>
<th>( \mathcal{M} \setminus {c} )</th>
<th>( \neg \text{SAT}(\mathcal{M} \setminus {c}) )</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_1..c_7 )</td>
<td>( c_2..c_7 )</td>
<td>1</td>
<td>Drop ( c_1 )</td>
</tr>
<tr>
<td>( c_2..c_7 )</td>
<td>( c_3..c_7 )</td>
<td>1</td>
<td>Drop ( c_2 )</td>
</tr>
<tr>
<td>( c_3..c_7 )</td>
<td>( c_4..c_7 )</td>
<td>1</td>
<td>Drop ( c_3 )</td>
</tr>
<tr>
<td>( c_4..c_7 )</td>
<td>( c_5..c_7 )</td>
<td>0</td>
<td>Keep ( c_4 )</td>
</tr>
<tr>
<td>( c_4..c_7 )</td>
<td>( c_4c_6c_7 )</td>
<td>0</td>
<td>Keep ( c_5 )</td>
</tr>
<tr>
<td>( c_4..c_7 )</td>
<td>( c_4c_5c_7 )</td>
<td>0</td>
<td>Keep ( c_6 )</td>
</tr>
<tr>
<td>( c_4..c_7 )</td>
<td>( c_4..c_6 )</td>
<td>1</td>
<td>Drop ( c_7 )</td>
</tr>
</tbody>
</table>

- MUS: \( \{c_4, c_5, c_6\} \)
Many MUS algorithms

- Formula $\mathcal{F}$ with $m$ clauses $k$ the size of largest minimal subset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Oracle Calls</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion-based</td>
<td>$O(km)$</td>
<td>[dSNP88, vMW08]</td>
</tr>
<tr>
<td>MCS_MUS</td>
<td>$O(km)$</td>
<td>[BK15]</td>
</tr>
<tr>
<td>Deletion-based</td>
<td>$O(m)$</td>
<td>[CD91, BDTW93]</td>
</tr>
<tr>
<td>Linear insertion</td>
<td>$O(m)$</td>
<td>[MSL11, BLM12]</td>
</tr>
<tr>
<td>Dichotomic</td>
<td>$O(k \log(m))$</td>
<td>[HLSB06]</td>
</tr>
<tr>
<td>QuickXplain</td>
<td>$O(k + k \log(\frac{m}{k}))$</td>
<td>[Jun04]</td>
</tr>
<tr>
<td>Progression</td>
<td>$O(k \log(1 + \frac{m}{k}))$</td>
<td>[MJB13]</td>
</tr>
</tbody>
</table>

- **Note:** Lower bound in $\mathsf{FP}^\mathsf{NP}_{113}$ and upper bound in $\mathsf{FP}^\mathsf{NP}$ [CT95]
- Oracle calls correspond to testing unsatisfiability with SAT solver
- Practical optimizations: clause set trimming; clause set refinement; redundancy removal; (recursive) model rotation
Outline

Minimal Unsatisfiability

Maximum Satisfiability

Examples in PySAT
Recap MaxSAT

- Given **unsatisfiable** formula, find **largest** subset of clauses that is satisfiable
Recap MaxSAT

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (MCS) is an irreducible relaxation of the formula
Recap MaxSAT

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (MCS) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes
Recap MaxSAT

Given unsatisfiable formula, find largest subset of clauses that is satisfiable

A Minimal Correction Subset (MCS) is an irreducible relaxation of the formula

The MaxSAT solution is one of the smallest MCSes

- **Note**: Clauses can have weights & there can be hard clauses
Recap MaxSAT

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (MCS) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest cost MCSes
  - Note: Clauses can have weights & there can be hard clauses
Recap MaxSAT

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (MCS) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest cost MCSes
  - Note: Clauses can have weights & there can be hard clauses
- Many practical applications
MaxSAT problem(s)

<table>
<thead>
<tr>
<th>Hard Clauses?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights?</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Weights?</td>
<td>Hard Clauses?</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>---------------</td>
<td>---</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Plain</td>
</tr>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>Weighted</td>
<td>Weighted Partial</td>
</tr>
</tbody>
</table>
MaxSAT problem(s)

<table>
<thead>
<tr>
<th>Hard Clauses?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights?</td>
<td>No</td>
<td>Plain</td>
</tr>
<tr>
<td>No</td>
<td>Plain</td>
<td>Partial</td>
</tr>
<tr>
<td>Yes</td>
<td>Weighted</td>
<td>Weighted Partial</td>
</tr>
</tbody>
</table>

- **Must** satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost
  (s.t. hard & remaining soft clauses are satisfied)
### MaxSAT problem(s)

<table>
<thead>
<tr>
<th>Hard Clauses?</th>
<th>No</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weights?</td>
<td>No</td>
<td>Plain</td>
</tr>
<tr>
<td>No</td>
<td>Plain</td>
<td>Partial</td>
</tr>
<tr>
<td>Yes</td>
<td>Weighted</td>
<td>Weighted Partial</td>
</tr>
</tbody>
</table>

- **Must** satisfy hard clauses, if any
- Compute set of satisfied soft clauses with **maximum cost**
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with **minimum cost**
  (s.t. hard & remaining soft clauses are satisfied)

- **Note**: goal is to compute set of satisfied (or falsified) clauses; **not** just the cost!
Issues with MaxSAT

- Unit propagation is unsound for MaxSAT

\[
\begin{align*}
\text{Formula with all clauses soft: } & \{ (x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z) \} \\
\text{After unit propagation: } & \{ (x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z) \} \\
\text{Is 2 the MaxSAT solution? } & \text{No! Enough to either falsify } (x) \text{ or } (z)
\end{align*}
\]
Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
• **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:
    \[
    \{ (x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z) \} \]
  - After unit propagation:
    \[
    \{ (x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z) \} \]
Unit propagation is unsound for MaxSAT

- Formula with all clauses soft:
  \[
  \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}\n  \]

- After unit propagation:
  \[
  \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}\n  \]

- Is 2 the MaxSAT solution??
Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - After unit propagation:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - Is 2 the MaxSAT solution??
  - **No!** Enough to either falsify (x) or (z)
Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}\]
  - After unit propagation:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}\]
  - Is 2 the MaxSAT solution??
    - **No!** Enough to either falsify \((x)\) or \((z)\)

- **Cannot** use unit propagation
Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - After unit propagation:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - Is 2 the MaxSAT solution??
    - **No!** Enough to either falsify \((x)\) or \((z)\)

- **Cannot** use unit propagation
- **Cannot** learn clauses (using unit propagation)
Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - After unit propagation:
    \[
    \{(x), (\neg x \lor y_1), (\neg x \lor y_2), (\neg y_1 \lor \neg z), (\neg y_2 \lor \neg z), (z)\}
    \]
  - Is 2 the MaxSAT solution??
    - **No!** Enough to either falsify \((x)\) or \((z)\)

- **Cannot** use unit propagation
- **Cannot** learn clauses (using unit propagation)
- Need to solve MaxSAT using different techniques
Many MaxSAT approaches

MaxSAT Algorithms

- Branch & Bound
  - No unit prop; No cl. learning

- Core Guided
  - Relax cls given unsat cores

- Model Guided
  - Relax cls given models

- Iterative
  - All cls relaxed

- Iterative MHS & SAT
  - Iterative MHS

For practical (industrial) instances: core-guided & iterative MHS approaches are the most effective [MaxSAT14].
Many MaxSAT approaches

- For practical (industrial) instances: core-guided & iterative MHS approaches are the most effective

[MaxSAT14]
Core-guided solver performance – partial

Number x of instances solved in y seconds

CPU time in seconds
Number of instances
Number x of instances solved in y seconds
Open-WBO-In
QMaxSAT2-mt-13
QMaxSat-g2-12
QMaxSat0.4-11
QMaxSat-10

Source: [MaxSAT 2014 organizers]
Core-guided solver performance – weighted partial

Number x of instances solved in y seconds

Source: [MaxSAT 2014 organizers]
Outline

Minimal Unsatisfiability

Maximum Satisfiability
  Iterative SAT Solving
  Core-Guided Algorithms
  Minimum Hitting Sets

Examples in PySAT
Basic MaxSAT with iterative SAT solving

\[
\begin{align*}
&x_6 \lor x_2 & \neg x_6 \lor x_2 & \neg x_2 \lor x_1 & \neg x_1 \\
&\neg x_6 \lor x_8 & x_6 \lor \neg x_8 & x_2 \lor x_4 & \neg x_4 \lor x_5 \\
&x_7 \lor x_5 & \neg x_7 \lor x_5 & \neg x_5 \lor x_3 & \neg x_3
\end{align*}
\]

Example CNF formula
Basic MaxSAT with iterative SAT solving

\[ x_6 \lor x_2 \lor r_1 \quad \neg x_6 \lor x_2 \lor r_2 \quad \neg x_2 \lor x_1 \lor r_3 \quad \neg x_1 \lor r_4 \]

\[ \neg x_6 \lor x_8 \lor r_5 \quad x_6 \lor \neg x_8 \lor r_6 \quad x_2 \lor x_4 \lor r_7 \quad \neg x_4 \lor x_5 \lor r_8 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_{11} \quad \neg x_3 \lor r_{12} \]

\[ \sum_{i=1}^{12} r_i \leq 12 \]

Relax all clauses; Set \( UB = 12 + 1 \)
Basic MaxSAT with iterative SAT solving

\[
\begin{align*}
    &x_6 \lor x_2 \lor r_1 & \neg x_6 \lor x_2 \lor r_2 & \neg x_2 \lor x_1 \lor r_3 & \neg x_1 \lor r_4 \\
    &\neg x_6 \lor x_8 \lor r_5 & x_6 \lor \neg x_8 \lor r_6 & x_2 \lor x_4 \lor r_7 & \neg x_4 \lor x_5 \lor r_8 \\
    &x_7 \lor x_5 \lor r_9 & \neg x_7 \lor x_5 \lor r_{10} & \neg x_5 \lor x_3 \lor r_{11} & \neg x_3 \lor r_{12}
\end{align*}
\]

\[\sum_{i=1}^{12} r_i \leq 12\]

Formula is SAT; E.g. all \(x_i = 0\) and \(r_1 = r_7 = r_9 = 1\) (i.e. cost = 3)
Basic MaxSAT with iterative SAT solving

\[ x_6 \lor x_2 \lor r_1 \quad \neg x_6 \lor x_2 \lor r_2 \quad \neg x_2 \lor x_1 \lor r_3 \quad \neg x_1 \lor r_4 \]

\[ \neg x_6 \lor x_8 \lor r_5 \quad x_6 \lor \neg x_8 \lor r_6 \quad x_2 \lor x_4 \lor r_7 \quad \neg x_4 \lor x_5 \lor r_8 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_{11} \quad \neg x_3 \lor r_{12} \]

\[ \sum_{i=1}^{12} r_i \leq 2 \]

Refine \( UB = 3 \)
Basic MaxSAT with iterative SAT solving

\[ x_6 \lor x_2 \lor r_1 \quad \neg x_6 \lor x_2 \lor r_2 \quad \neg x_2 \lor x_1 \lor r_3 \quad \neg x_1 \lor r_4 \]

\[-x_6 \lor x_8 \lor r_5 \quad x_6 \lor \neg x_8 \lor r_6 \quad x_2 \lor x_4 \lor r_7 \quad \neg x_4 \lor x_5 \lor r_8 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_{11} \quad \neg x_3 \lor r_{12} \]

\[ \sum_{i=1}^{12} r_i \leq 2 \]

Formula is **SAT**; E.g. \( x_1 = x_2 = 1; x_3 = \ldots = x_8 = 0 \) and \( r_4 = r_9 = 1 \)

(i.e. cost = 2)
Basic MaxSAT with iterative SAT solving

\[
\begin{align*}
& x_6 \lor x_2 \lor r_1 & \neg x_6 \lor x_2 \lor r_2 & \neg x_2 \lor x_1 \lor r_3 & \neg x_1 \lor r_4 \\
& \neg x_6 \lor x_8 \lor r_5 & x_6 \lor \neg x_8 \lor r_6 & x_2 \lor x_4 \lor r_7 & \neg x_4 \lor x_5 \lor r_8 \\
& x_7 \lor x_5 \lor r_9 & \neg x_7 \lor x_5 \lor r_{10} & \neg x_5 \lor x_3 \lor r_{11} & \neg x_3 \lor r_{12} \\
\sum_{i=1}^{12} r_i & \leq 1
\end{align*}
\]

Refine \( UB = 2 \)
Basic MaxSAT with iterative SAT solving

\[x_6 \lor x_2 \lor r_1\]
\[\neg x_6 \lor x_2 \lor r_2\]
\[\neg x_2 \lor x_1 \lor r_3\]
\[\neg x_1 \lor r_4\]
\[\neg x_6 \lor x_8 \lor r_5\]
\[x_6 \lor \neg x_8 \lor r_6\]
\[x_2 \lor x_4 \lor r_7\]
\[\neg x_4 \lor x_5 \lor r_8\]
\[x_7 \lor x_5 \lor r_9\]
\[\neg x_7 \lor x_5 \lor r_{10}\]
\[\neg x_5 \lor x_3 \lor r_{11}\]
\[\neg x_3 \lor r_{12}\]
\[
\sum_{i=1}^{12} r_i \leq 1
\]

Formula is UNSAT; terminate
Basic MaxSAT with iterative SAT solving

\[ x_6 \lor x_2 \lor r_1 \quad \neg x_6 \lor x_2 \lor r_2 \quad \neg x_2 \lor x_1 \lor r_3 \quad \neg x_1 \lor r_4 \]

\[ \neg x_6 \lor x_8 \lor r_5 \quad x_6 \lor \neg x_8 \lor r_6 \quad x_2 \lor x_4 \lor r_7 \quad \neg x_4 \lor x_5 \lor r_8 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_{11} \quad \neg x_3 \lor r_{12} \]

\[ \sum_{i=1}^{12} r_i \leq 1 \]

MaxSAT solution is last satisfied UB: \( UB = 2 \)
Basic MaxSAT with iterative SAT solving

\[ x_6 \lor x_2 \lor r_1 \quad \neg x_6 \lor x_2 \lor r_2 \quad \neg x_2 \lor x_1 \lor r_3 \quad \neg x_1 \lor r_4 \]
\[ \neg x_6 \lor x_8 \lor r_5 \quad x_6 \lor \neg x_8 \lor r_6 \quad x_2 \lor x_4 \lor r_7 \quad \neg x_4 \lor x_5 \lor r_8 \]
\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_{11} \quad \neg x_3 \lor r_{12} \]
\[ \sum_{i=1}^{12} r_i \leq 1 \]

MaxSAT solution is last satisfied UB: \( UB = 2 \)

AtMost\( k \)/PB constraints over all relaxation variables

All (possibly many) soft clauses relaxed
Outline

Minimal Unsatisfiability

Maximum Satisfiability
  Iterative SAT Solving
  Core-Guided Algorithms
  Minimum Hitting Sets

Examples in PySAT
Example CNF formula
MSU3 core-guided algorithm

\[ x_6 \lor x_2 \quad \neg x_6 \lor x_2 \]
\[ x_6 \lor \neg x_8 \quad x_6 \lor \neg x_8 \]
\[ x_7 \lor x_5 \quad \neg x_7 \lor x_5 \]
\[ \neg x_5 \lor x_3 \quad \neg x_3 \]

Formula is \textbf{UNSAT}; \textbf{OPT} \leq |\varphi| - 1; Get unsat core
MSU3 core-guided algorithm

\[ x_6 \lor x_2 \quad \neg x_6 \lor x_2 \quad \neg x_2 \lor x_1 \lor r_1 \quad \neg x_1 \lor r_2 \]

\[ \neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \quad x_2 \lor x_4 \lor r_3 \quad \neg x_4 \lor x_5 \lor r_4 \]

\[ x_7 \lor x_5 \quad \neg x_7 \lor x_5 \quad \neg x_5 \lor x_3 \lor r_5 \quad \neg x_3 \lor r_6 \]

\[ \sum_{i=1}^{6} r_i \leq 1 \]

Add relaxation variables and AtMost \( k \), \( k = 1 \), constraint
MSU3 core-guided algorithm

Formula is (again) **UNSAT**; $\text{OPT} \leq |\varphi| - 2$; Get unsat core
MSU3 core-guided algorithm

\[
x_6 \lor x_2 \lor r_7 \quad \neg x_6 \lor x_2 \lor r_8 \quad \neg x_2 \lor x_1 \lor r_1 \quad \neg x_1 \lor r_2
\]

\[
\neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \quad x_2 \lor x_4 \lor r_3 \quad \neg x_4 \lor x_5 \lor r_4
\]

\[
x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_5 \quad \neg x_3 \lor r_6
\]

\[
\sum_{i=1}^{10} r_i \leq 2
\]

Add new relaxation variables and update AtMost \( k \), \( k=2 \), constraint
MSU3 core-guided algorithm

\[ x_6 \lor x_2 \lor r_7 \quad \neg x_6 \lor x_2 \lor r_8 \quad \neg x_2 \lor x_1 \lor r_1 \quad \neg x_1 \lor r_2 \]

\[ \neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \quad x_2 \lor x_4 \lor r_3 \quad \neg x_4 \lor x_5 \lor r_4 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_5 \quad \neg x_3 \lor r_6 \]

\[ \sum_{i=1}^{10} r_i \leq 2 \]

Instance is now SAT
MSU3 core-guided algorithm

\[
\begin{align*}
    x_6 \lor x_2 \lor r_7 & \quad \neg x_6 \lor x_2 \lor r_8 & \quad \neg x_2 \lor x_1 \lor r_1 & \quad \neg x_1 \lor r_2 \\
    \neg x_6 \lor x_8 & \quad x_6 \lor \neg x_8 & \quad x_2 \lor x_4 \lor r_3 & \quad \neg x_4 \lor x_5 \lor r_4 \\
    x_7 \lor x_5 \lor r_9 & \quad \neg x_7 \lor x_5 \lor r_{10} & \quad \neg x_5 \lor x_3 \lor r_5 & \quad \neg x_3 \lor r_6 \\
\end{align*}
\]

\[\sum_{i=1}^{10} r_i \leq 2\]

MaxSAT solution is \(|\varphi| - \mathcal{I} = 12 - 2 = 10\)
MSU3 core-guided algorithm

\[
\begin{align*}
x_6 \lor x_2 \lor r_7 & \quad \neg x_6 \lor x_2 \lor r_8 & \quad \neg x_2 \lor x_1 \lor r_1 & \quad \neg x_1 \lor r_2 \\
\neg x_6 \lor x_8 & \quad x_6 \lor \neg x_8 & \quad x_2 \lor x_4 \lor r_3 & \quad \neg x_4 \lor x_5 \lor r_4 \\
x_7 \lor x_5 \lor r_9 & \quad \neg x_7 \lor x_5 \lor r_{10} & \quad \neg x_5 \lor x_3 \lor r_5 & \quad \neg x_3 \lor r_6 \\
\sum_{i=1}^{10} r_i \leq 2
\end{align*}
\]

MaxSAT solution is \(|\varphi| - I = 12 - 2 = 10\)

AtMost k/PB constraints used

Relaxed soft clauses become hard
MSU3 core-guided algorithm

\[ x_6 \lor x_2 \lor r_7 \quad \neg x_6 \lor x_2 \lor r_8 \quad \neg x_2 \lor x_1 \lor r_1 \quad \neg x_1 \lor r_2 \]

\[ \neg x_6 \lor x_8 \quad x_6 \lor \neg x_8 \quad x_2 \lor x_4 \lor r_3 \quad \neg x_4 \lor x_5 \lor r_4 \]

\[ x_7 \lor x_5 \lor r_9 \quad \neg x_7 \lor x_5 \lor r_{10} \quad \neg x_5 \lor x_3 \lor r_5 \quad \neg x_3 \lor r_6 \]

\[ \sum_{i=1}^{10} r_i \leq 2 \]

MaxSAT solution is \(|\varphi| - I = 12 - 2 = 10\)

AtMostk/PB constraints used

Some clauses not relaxed

Relaxed soft clauses become hard
Outline

Minimal Unsatisfiability

**Maximum Satisfiability**
- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

Examples in PySAT
MHS approach for MaxSAT

\begin{align*}
c_1 &= x_6 \lor x_2 & c_2 &= \neg x_6 \lor x_2 & c_3 &= \neg x_2 \lor x_1 & c_4 &= \neg x_1 \\
c_5 &= \neg x_6 \lor x_8 & c_6 &= x_6 \lor \neg x_8 & c_7 &= x_2 \lor x_4 & c_8 &= \neg x_4 \lor x_5 \\
c_9 &= x_7 \lor x_5 & c_{10} &= \neg x_7 \lor x_5 & c_{11} &= \neg x_5 \lor x_3 & c_{12} &= \neg x_3
\end{align*}

\mathcal{K} = \emptyset

- Find MHS of \mathcal{K}:
MHS approach for MaxSAT

\[c_1 = x_6 \lor x_2, \quad c_2 = \neg x_6 \lor x_2, \quad c_3 = \neg x_2 \lor x_1, \quad c_4 = \neg x_1\]

\[c_5 = \neg x_6 \lor x_8, \quad c_6 = x_6 \lor \neg x_8, \quad c_7 = x_2 \lor x_4, \quad c_8 = \neg x_4 \lor x_5\]

\[c_9 = x_7 \lor x_5, \quad c_{10} = \neg x_7 \lor x_5, \quad c_{11} = \neg x_5 \lor x_3, \quad c_{12} = \neg x_3\]

\[\mathcal{K} = \emptyset\]

- Find MHS of \(\mathcal{K}\): \(\emptyset\)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \emptyset \]

- Find MHS of \( \mathcal{K} \): \( \emptyset \)
- \( \text{SAT}(\mathcal{F} \setminus \emptyset) \)?
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \emptyset \]

- Find MHS of \( \mathcal{K} \): \( \emptyset \)
- \( \text{SAT}(\mathcal{F} \setminus \emptyset) \)? No
MHS approach for MaxSAT

\[
\begin{align*}
c_1 &= x_6 \lor x_2 \quad & c_2 &= \neg x_6 \lor x_2 \quad & c_3 &= \neg x_2 \lor x_1 \quad & c_4 &= \neg x_1 \\
c_5 &= \neg x_6 \lor x_8 \quad & c_6 &= x_6 \lor \neg x_8 \quad & c_7 &= x_2 \lor x_4 \quad & c_8 &= \neg x_4 \lor x_5 \\
c_9 &= x_7 \lor x_5 \quad & c_{10} &= \neg x_7 \lor x_5 \quad & c_{11} &= \neg x_5 \lor x_3 \quad & c_{12} &= \neg x_3
\end{align*}
\]

\[K = \emptyset\]

- Find MHS of \(K\): \(\emptyset\)
- \(\text{SAT}(\mathcal{F} \setminus \emptyset)\)? No
- Core of \(\mathcal{F}\): \(\{c_1, c_2, c_3, c_4\}\)
MHS approach for MaxSAT

\[
\begin{align*}
    c_1 &= x_6 \lor x_2 \\
    c_2 &= \neg x_6 \lor x_2 \\
    c_3 &= \neg x_2 \lor x_1 \\
    c_4 &= \neg x_1 \\
    c_5 &= \neg x_6 \lor x_8 \\
    c_6 &= x_6 \lor \neg x_8 \\
    c_7 &= x_2 \lor x_4 \\
    c_8 &= \neg x_4 \lor x_5 \\
    c_9 &= x_7 \lor x_5 \\
    c_{10} &= \neg x_7 \lor x_5 \\
    c_{11} &= \neg x_5 \lor x_3 \\
    c_{12} &= \neg x_3
\end{align*}
\]

\[\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}\]

- Find MHS of \(\mathcal{K}\): \(\emptyset\)
- \(\text{SAT}(\mathcal{F} \setminus \emptyset)\)? No
- Core of \(\mathcal{F}\): \(\{c_1, c_2, c_3, c_4\}\). Update \(\mathcal{K}\)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{ \{ c_1, c_2, c_3, c_4 \} \} \]

- Find MHS of \( \mathcal{K} \):
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_1\} \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ K = \{ \{ c_1, c_2, c_3, c_4 \} \} \]

- Find MHS of \( K \): E.g. \( \{ c_1 \} \)
- \( \text{SAT}(F \setminus \{ c_1 \})? \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad \quad c_2 = \neg x_6 \lor x_2 \quad \quad c_3 = \neg x_2 \lor x_1 \quad \quad c_4 = \neg x_1 \]
\[ c_5 = \neg x_6 \lor x_8 \quad \quad c_6 = x_6 \lor \neg x_8 \quad \quad c_7 = x_2 \lor x_4 \quad \quad c_8 = \neg x_4 \lor x_5 \]
\[ c_9 = x_7 \lor x_5 \quad \quad c_{10} = \neg x_7 \lor x_5 \quad \quad c_{11} = \neg x_5 \lor x_3 \quad \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_1\} \)
- \( \text{SAT}(\mathcal{F} \setminus \{c_1\})? \) No
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ K = \{c_1, c_2, c_3, c_4\} \]

- Find MHS of \( K \): E.g. \( \{c_1\} \)
- \( \text{SAT}(\mathcal{F} \setminus \{c_1\}) \)? No
- Core of \( \mathcal{F} \): \( \{c_9, c_{10}, c_{11}, c_{12}\} \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \\{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_1\} \)
- \( SAT(\mathcal{F} \setminus \{c_1\})? \) No
- Core of \( \mathcal{F} \): \( \{c_9, c_{10}, c_{11}, c_{12}\} \). Update \( \mathcal{K} \)
MHS approach for MaxSAT

\[\begin{align*}
c_1 &= x_6 \lor x_2 \\
c_2 &= \neg x_6 \lor x_2 \\
c_3 &= \neg x_2 \lor x_1 \\
c_4 &= \neg x_1 \\
c_5 &= \neg x_6 \lor x_8 \\
c_6 &= x_6 \lor \neg x_8 \\
c_7 &= x_2 \lor x_4 \\
c_8 &= \neg x_4 \lor x_5 \\
c_9 &= x_7 \lor x_5 \\
c_{10} &= \neg x_7 \lor x_5 \\
c_{11} &= \neg x_5 \lor x_3 \\
c_{12} &= \neg x_3
\end{align*}\]

\[\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}\]

- Find MHS of \(\mathcal{K}\):
MHS approach for MaxSAT

\[
c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \\
c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \\
c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3
\]

\[\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}\]

- Find MHS of \(\mathcal{K}\): E.g. \(\{c_1, c_9\}\)
MHS approach for MaxSAT

\begin{align*}
    c_1 &= x_6 \vee x_2 &
    c_2 &= \neg x_6 \vee x_2 &
    c_3 &= \neg x_2 \vee x_1 &
    c_4 &= \neg x_1 \\
    c_5 &= \neg x_6 \vee x_8 &
    c_6 &= x_6 \vee \neg x_8 &
    c_7 &= x_2 \vee x_4 &
    c_8 &= \neg x_4 \vee x_5 \\
    c_9 &= x_7 \vee x_5 &
    c_{10} &= \neg x_7 \vee x_5 &
    c_{11} &= \neg x_5 \vee x_3 &
    c_{12} &= \neg x_3 \\
\end{align*}

\[ \mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_1, c_9\} \)
- \( \text{SAT}(\mathcal{F} \setminus \{c_1, c_9\}) \)?
MHS approach for MaxSAT

\[
\begin{align*}
    c_1 &= x_6 \lor x_2 & c_2 &= \neg x_6 \lor x_2 & c_3 &= \neg x_2 \lor x_1 & c_4 &= \neg x_1 \\
    c_5 &= \neg x_6 \lor x_8 & c_6 &= x_6 \lor \neg x_8 & c_7 &= x_2 \lor x_4 & c_8 &= \neg x_4 \lor x_5 \\
    c_9 &= x_7 \lor x_5 & c_{10} &= \neg x_7 \lor x_5 & c_{11} &= \neg x_5 \lor x_3 & c_{12} &= \neg x_3
\end{align*}
\]

\[\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}\]

- Find MHS of \(\mathcal{K}\): E.g. \(\{c_1, c_9\}\)
- \(\text{SAT}(\mathcal{F} \setminus \{c_1, c_9\})\)? No
MHS approach for MaxSAT

\[
\begin{align*}
    c_1 &= x_6 \vee x_2 &
    c_2 &= \neg x_6 \vee x_2 &
    c_3 &= \neg x_2 \vee x_1 &
    c_4 &= \neg x_1 \\
    c_5 &= \neg x_6 \vee x_8 &
    c_6 &= x_6 \vee \neg x_8 &
    c_7 &= x_2 \vee x_4 &
    c_8 &= \neg x_4 \vee x_5 \\
    c_9 &= x_7 \vee x_5 &
    c_{10} &= \neg x_7 \vee x_5 &
    c_{11} &= \neg x_5 \vee x_3 &
    c_{12} &= \neg x_3
\end{align*}
\]

\[K = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}\]

- Find MHS of \(K\): E.g. \(\{c_1, c_9\}\)
- \(\text{SAT}(\mathcal{F} \setminus \{c_1, c_9\})? \quad \text{No}\)
- Core of \(\mathcal{F}\): \(\{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{ \{ c_1, c_2, c_3, c_4 \}, \{ c_9, c_{10}, c_{11}, c_{12} \}, \{ c_3, c_4, c_7, c_8, c_{11}, c_{12} \} \} \]

- Find MHS of \( \mathcal{K} \): E.g. \{c_1, c_9\}
- \( \text{SAT}(\mathcal{F} \setminus \{c_1, c_9\}) \)\? No
- Core of \( \mathcal{F} \): \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}. Update \( \mathcal{K} \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad \quad c_2 = \neg x_6 \lor x_2 \quad \quad c_3 = \neg x_2 \lor x_1 \quad \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad \quad c_6 = x_6 \lor \neg x_8 \quad \quad c_7 = x_2 \lor x_4 \quad \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad \quad c_{10} = \neg x_7 \lor x_5 \quad \quad c_{11} = \neg x_5 \lor x_3 \quad \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{ \{ c_1, c_2, c_3, c_4 \}, \{ c_9, c_{10}, c_{11}, c_{12} \}, \{ c_3, c_4, c_7, c_8, c_{11}, c_{12} \} \} \]

- Find MHS of \( \mathcal{K} \):
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \left\{ \{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\} \right\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_4, c_9\} \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \( \{c_4, c_9\} \)
- \( \text{SAT}(\mathcal{F} \setminus \{c_4, c_9\})? \)
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]
\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]
\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{ \{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\} \]

- **Find MHS of \( \mathcal{K} \):** E.g. \( \{c_4, c_9\} \)
- **SAT(\( \mathcal{F} \setminus \{c_4, c_9\} \))**? Yes
MHS approach for MaxSAT

\[ c_1 = x_6 \lor x_2 \quad c_2 = \neg x_6 \lor x_2 \quad c_3 = \neg x_2 \lor x_1 \quad c_4 = \neg x_1 \]

\[ c_5 = \neg x_6 \lor x_8 \quad c_6 = x_6 \lor \neg x_8 \quad c_7 = x_2 \lor x_4 \quad c_8 = \neg x_4 \lor x_5 \]

\[ c_9 = x_7 \lor x_5 \quad c_{10} = \neg x_7 \lor x_5 \quad c_{11} = \neg x_5 \lor x_3 \quad c_{12} = \neg x_3 \]

\[ \mathcal{K} = \{ \{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\} \]

- Find MHS of \( \mathcal{K} \): E.g. \{c_4, c_9\}
- \( SAT(\mathcal{F} \setminus \{c_4, c_9\})? \) Yes
- Terminate & return 2
MaxSAT solving with SAT oracles – a sample

- A sample of recent algorithms:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th># Oracle Queries</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear search SU</td>
<td>Exponential***</td>
<td>[BP10]</td>
</tr>
<tr>
<td>Binary search</td>
<td>Linear*</td>
<td>[FM06]</td>
</tr>
<tr>
<td>FM/WMSU1/WPM1</td>
<td>Exponential**</td>
<td>[FM06, MP08, MMSP09, ABL09, ABGL12]</td>
</tr>
<tr>
<td>WPM2</td>
<td>Exponential**</td>
<td>[ABL10a, ABL13]</td>
</tr>
<tr>
<td>Bin-Core-Dis</td>
<td>Linear</td>
<td>[HMM11, MHM12]</td>
</tr>
<tr>
<td>Iterative MHS</td>
<td>Exponential</td>
<td>[DB11, DB13a, DB13b]</td>
</tr>
</tbody>
</table>

* $O(\log m)$ queries with SAT oracle, for (partial) unweighted MaxSAT
** Weighted case; depends on computed cores
*** On # bits of problem instance (due to weights)

- But also additional recent work:
  - Progression
  - Soft cardinality constraints (OLL)
  - MaxSAT resolution
  - ...

[BP10]  
[FM06]  
[FM06, MP08, MMSP09, ABL09, ABGL12]  
[ABL10a, ABL13]  
[HMM11, MHM12]  
[DB11, DB13a, DB13b]
Outline

Minimal Unsatisfiability

Maximum Satisfiability

Examples in PySAT
Example: naive (deletion) MUS extraction

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$

begin

$\mathcal{M} \leftarrow \mathcal{F}$

foreach $c \in \mathcal{M}$ do

if $\neg$SAT($\mathcal{M} \setminus \{c\}$) then

$\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$

// If $\neg$SAT($\mathcal{M} \setminus \{c\}$), then $c \notin$ MUS

return $\mathcal{M}$

// Final $\mathcal{M}$ is MUS

end

- Number of predicate tests: $O(m)$

[CD91, BDTW93]
def main():
    cnf = CNF(from_file=argv[1])  # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)

    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)

    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)

if __name__ == "__main__":
    main()
def add_assumps(cnf):
    rnv = topv = cnf.nv
    assumps = [] # list of assumptions to use
    for i in range(len(cnf.clauses)):
        topv += 1
        assumps.append(topv) # register literal
        cnf.clauses[i].append(-topv) # extend clause with literal
    cnf.nv = cnf.nv + len(assumps) # update # of vars
    return rnv, assumps

def main():
    cnf = CNF(from_file=argv[1]) # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)
    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)
    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)
    if __name__ == "__main__":
        main()
from sys import argv

from pysat.formula import CNF
from pysat.solvers import Solver

def find_mus(assmp, oracle):
    i = 0
    while i < len(assmp):
        ts = assmp[:i] + assmp[(i+1):]
        if not oracle.solve(assumptions=ts):
            assmp = ts
        else:
            i += 1
    return assmp
from sys import argv
from pysat.formula import CNF
from pysat.solvers import Solver

def find_mus(assmp, oracle):
    i = 0
    while i < len(assmp):
        ts = assmp[:i] + assmp[(i+1):]
        if not oracle.solve(assumptions=ts):
            assmp = ts
        else:
            i += 1
    return assmp

Demo
A less naive MUS extractor

```python
def clset_refine(assmp, oracle):
    assmp = sorted(assmp)
    while True:
        oracle.solve(assumptions=assmp)
        ts = sorted(oracle.get_core())
        if ts == assmp:
            break
        assmp = ts
    return assmp

# ...
def main():
    cnf = CNF(from_file=argv[1])  # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)

    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)

    assumps = clset_refine(assumps, oracle)
    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)

    if __name__ == "__main__":
        main()
```

Encoding sudoku

class SudokuEncoding(CNF, object):
    def __init__(self):
        # initializing CNF's internal parameters
        super(SudokuEncoding, self).__init__()
        self.vpool = IDPool()
        # at least one value in each cell
        for i, j in itertools.product(range(9), range(9)):
            self.append([self.var(i, j, val) for val in range(9)])
        # at most one value in each row
        for i in range(9):
            for val in range(9):
                for j1, j2 in itertools.combinations(range(9), 2):
                    self.append([-self.var(i, j1, val), -self.var(i, j2, val)])
        # at most one value in each column
        for j in range(9):
            for val in range(9):
                for i1, i2 in itertools.combinations(range(9), 2):
                    self.append([-self.var(i1, j, val), -self.var(i2, j, val)])
        # at most one value in each square
        for val in range(9):
            for i in range(3):
                for j in range(3):
                    subgrid = itertools.product(range(3*i, 3*i+3), range(3*j, 3*j+3))
                    for c in itertools.combinations(subgrid, 2):
                        self.append([-self.var(c[0][0], c[0][1], val),
                                      -self.var(c[1][0], c[1][1], val)])

    def var(self, i, j, v):
        return self.vpool.id(tuple([i + 1, j + 1, v + 1]))

    def cell(self, var):
        return self.vpool.obj(var)
A prototype sudoku game
A prototype sudoku game
A prototype sudoku game

Demo
Part 4

Sample of Applications
Flagship applications

- Bounded (& unbounded) model checking
- Automated planning
- Software model checking
- Package management
- Design debugging
- Haplotyping
CDCL SAT is the engines’ engine

Engines using SAT engines

Boolean
- QBF
- MaxSAT
- PBO
- #SAT

FOL
- Theorem proving
- Model finding

Other
- ASP
- LCG
- CSP

...
CDCL SAT is ubiquitous in problem solving

- Problem Solving with SAT
  - Encodings
    - BMC
    - Planning
    - Eager SMT
    - MBD
  - Oracles
    - MC: ic3
    - CEGAR QBF
    - Counting
    - Enumeration
    - Min. Models
    - Backbones
    - MCS
    - MUS
    - MaxSAT
  - Embeddings
    - PBO
    - B&B Search
    - Enumeration
    - OPT SAT
    - Lazy SMT
    - LCG
Recent applications

- **Two-level logic minimization with SAT**
  - Reimplementation of Quine-McCluskey with SAT oracles

[IPM15]
Recent applications

- Two-level logic minimization with SAT
  - Reimplementation of Quine-McCluskey with SAT oracles

- Maximum cliques with SAT

- Explainable decision sets
  - Computation of smallest decision sets (rules)

- Smallest (explainable) decision trees
  - Computation of smallest decision trees

- Abduction-based explanations for ML models
  - On-demand extraction of explanations for any ML model
Recent applications

- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles

- **Maximum cliques with SAT** [IMM17]

- **Explainable decision sets** [IPNM18]
  - Computation of smallest decision sets (rules)
Recent applications

- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles

- **Maximum cliques with SAT** [IMM17]

- **Explainable decision sets** [IPNM18]
  - Computation of smallest decision sets (rules)

- **Smallest (explainable) decision trees** [NIPM18]
  - Computation of smallest decision trees
Recent applications

- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles

- **Maximum cliques with SAT** [IMM17]

- **Explainable decision sets** [IPNM18]
  - Computation of smallest decision sets (rules)

- **Smallest (explainable) decision trees** [NIPM18]
  - Computation of smallest decision trees

- **Abduction-based explanations for ML models** [INMS19]
  - On-demand extraction of explanations for any ML model
Smallest decision trees – encoding sizes in bytes

<table>
<thead>
<tr>
<th>Model</th>
<th>Weather</th>
<th>Mouse</th>
<th>Cancer</th>
<th>Car</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP’09*</td>
<td>27K</td>
<td>3.5M</td>
<td>92G</td>
<td>842M</td>
<td>354G</td>
</tr>
</tbody>
</table>

[NIPM18]
## Smallest decision trees – encoding sizes in bytes

<table>
<thead>
<tr>
<th>Model</th>
<th>Weather</th>
<th>Mouse</th>
<th>Cancer</th>
<th>Car</th>
<th>Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP’09*</td>
<td>27K</td>
<td>3.5M</td>
<td>92G</td>
<td>842M</td>
<td>354G</td>
</tr>
<tr>
<td>IJCAI’18</td>
<td>190K</td>
<td>1.2M</td>
<td>5.2M</td>
<td>4.1M</td>
<td>1.2G</td>
</tr>
</tbody>
</table>

[NIPM18]
Abduction-based explanations

• Positive:
  – General approach, applicable to any ML model represented as a set of constraints
  – Ability to explain predictions of NNs

• Negative:
  – NN sizes are fairly small, i.e. tens of neurons
  – Best results with ILP-based approach
    ▶ SMT/SAT models currently ineffective
    ▶ But, algorithms inspired SAT-based solutions
Solving MaxClique with SAT
Modeling MaxClique with SAT

- Given (undirected) graph, find largest complete subgraph

- Main constraint:

  Given $u, v \in V$:
  If $(u, v) \notin E$, then one must not have both $u$ and $v$ in the maximum-size clique
Modeling MaxClique with SAT

- Given (undirected) graph, find largest complete subgraph

- Main constraint:
  
  Given \( u, v \in V \):
  
  If \( (u, v) \notin E \), then one must not have both \( u \) and \( v \) in the maximum-size clique

- Associate Boolean \( x_u \) with \( u \in V \)
Modeling MaxClique with SAT

• Given (undirected) graph, find largest complete subgraph

• Main constraint:

  Given \( u, v \in V \):
  If \( (u, v) \notin E \), then one must not have both \( u \) and \( v \) in the maximum-size clique

• Associate Boolean \( x_u \) with \( u \in V \)

• Main goal:

  Assign 1 to largest set of variables that are consistent with constraint
  – E.g. use MaxSAT
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$

s.t. \[
\begin{align*}
\mathcal{H} & \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \} \\
S & \triangleq \{ (x_v) \mid v \in V \}
\end{align*}
\]

$H = \{ (\neg x_1 \lor \neg x_6) (\neg x_1 \lor \neg x_7) \\
(\neg x_2 \lor \neg x_6) (\neg x_2 \lor \neg x_7) \\
(\neg x_4 \lor \neg x_6) (\neg x_4 \lor \neg x_7) \\
(\neg x_6 \lor \neg x_7) \}
S = \{ (x_1) (x_2) (x_3) \\
(x_4) (x_5) (x_6) \\
(x_7) \}

solve $\mathcal{F}$ with MaxSAT !
An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t.

\[
\begin{align*}
\mathcal{H} & \triangleq \left\{ (-x_u \lor -x_v) \mid (u, v) \in E^C \right\} \\
\mathcal{S} & \triangleq \left\{ (x_u) \mid v \in V \right\}
\end{align*}
\]

solve $\mathcal{F}$ with MaxSAT!
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

$$
\mathcal{H} \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^c \}
$$

$$
S \triangleq \{ (x_u) \mid v \in V \}
$$

solve $\mathcal{F}$ with MaxSAT!
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

\[
\begin{align*}
\mathcal{H} & \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \} \\
S & \triangleq \{ (x_u) \mid v \in V \}
\end{align*}
\]

\begin{align*}
\mathcal{H} = & \{ \\
(\neg x_1 \lor \neg x_6) (\neg x_1 \lor \neg x_7) \\
(\neg x_2 \lor \neg x_6) (\neg x_2 \lor \neg x_7) \\
(\neg x_4 \lor \neg x_6) (\neg x_4 \lor \neg x_7) \\
(\neg x_6 \lor \neg x_7) \\
\}
\end{align*}

$S = \{ \\
(x_1) (x_2) (x_3) \\
(x_4) (x_5) (x_6) \\
(x_7) \\
\}$

solve $\mathcal{F}$ with MaxSAT !
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

\[
\mathcal{H} \triangleq \{ (\neg x_u \lor \neg x_v) | (u, v) \in E^c \} \\
S \triangleq \{ (x_u) | v \in V \}
\]

\[
\mathcal{H} = \left\{ \begin{array}{l}
(\neg x_1 \lor \neg x_6)(\neg x_1 \lor \neg x_7) \\
(\neg x_2 \lor \neg x_6)(\neg x_2 \lor \neg x_7) \\
(\neg x_4 \lor \neg x_6)(\neg x_4 \lor \neg x_7) \\
(\neg x_6 \lor \neg x_7)
\end{array} \right\}
\]

\[
S = \left\{ \begin{array}{l}
(x_1)(x_2)(x_3) \\
(x_4)(x_5)(x_6) \\
(x_7)
\end{array} \right\}
\]

solve $\mathcal{F}$ with MaxSAT!
An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t.

$$
\mathcal{H} \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \}
$$

$$
\mathcal{S} \triangleq \{ (x_u) \mid v \in V \}
$$

$$
H = \begin{cases} 
(\neg x_1 \lor \neg x_6) (\neg x_1 \lor \neg x_7) \\
(\neg x_2 \lor \neg x_6) (\neg x_2 \lor \neg x_7) \\
(\neg x_4 \lor \neg x_6) (\neg x_4 \lor \neg x_7) \\
(\neg x_6 \lor \neg x_7)
\end{cases}
$$

$$
S = \begin{cases} 
(x_1) (x_2) (x_3) \\
(x_4) (x_5) (x_6) \\
(x_7)
\end{cases}
$$

solve $\mathcal{F}$ with MaxSAT!
Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

$$
\begin{align*}
\mathcal{H} & \triangleq \left\{ (\neg u \lor \neg v) \mid (u, v) \in E^C \right\} \\
S & \triangleq \left\{ (x_u) \mid v \in V \right\}
\end{align*}
$$

\begin{align*}
\mathcal{H} & = \left\{ (\neg x_1 \lor \neg x_6)(\neg x_1 \lor \neg x_7) \\
& \quad (\neg x_2 \lor \neg x_6)(\neg x_2 \lor \neg x_7) \\
& \quad (\neg x_4 \lor \neg x_6)(\neg x_4 \lor \neg x_7) \\
& \quad (\neg x_6 \lor \neg x_7) \right\} \\
S & = \left\{ (x_1)(x_2)(x_3) \\
& \quad (x_4)(x_5)(x_6) \\
& \quad (x_7) \right\}
\end{align*}

solve $\mathcal{F}$ with MaxSAT!
Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

\[
\mathcal{H} \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \}
\]

\[
S \triangleq \{ (x_u) \mid v \in V \}
\]

solve $\mathcal{F}$ with MaxSAT!
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

\[
\begin{align*}
\mathcal{H} & \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \} \\
S & \triangleq \{ (x_u) \mid v \in V \}
\end{align*}
\]

\[
\mathcal{H} = \left\{ \begin{array}{l}
(\neg x_1 \lor \neg x_6)(\neg x_1 \lor \neg x_7) \\
(\neg x_2 \lor \neg x_6)(\neg x_2 \lor \neg x_7) \\
(\neg x_4 \lor \neg x_6)(\neg x_4 \lor \neg x_7) \\
(\neg x_6 \lor \neg x_7)
\end{array} \right\}
\]

\[
S = \left\{ \begin{array}{l}
(x_1)(x_2)(x_3) \\
(x_4)(x_5)(x_6) \\
(x_7)
\end{array} \right\}
\]

solve $\mathcal{F}$ with MaxSAT!
An example

Construct $\mathcal{F} = \langle \mathcal{H}, S \rangle$ s.t.

\[
\begin{align*}
\mathcal{H} & \triangleq \{ (\neg x_u \lor \neg x_v) \mid (u, v) \in E^C \} \\
S & \triangleq \{ (x_u) \mid v \in V \}
\end{align*}
\]

$H = \begin{cases} 
\neg x_1 \lor \neg x_6 & \neg x_1 \lor \neg x_7 \\
\neg x_2 \lor \neg x_6 & \neg x_2 \lor \neg x_7 \\
\neg x_4 \lor \neg x_6 & \neg x_4 \lor \neg x_7 \\
\neg x_6 \lor \neg x_7
\end{cases}$

$S = \begin{cases} 
(x_1)(x_2)(x_3) \\
(x_4)(x_5)(x_6) \\
(x_7)
\end{cases}$

solve $\mathcal{F}$ with MaxSAT!
But the size of $E^C$ can be **problematic**...

| Instance                        | $|V|$ | $|E|$  | $|E|^C$   |
|---------------------------------|-----|-------|----------|
| comm-n10000                      | 10000 | 10000 | 49995000 |
| ca-AstroPh                       | 18772 | 396160 | 175807218 |
| ca-citeseer                      | 227322 | 814136 | 25836945367 |
| ca-coauthors-dblp                | 540488 | 15245731 | 146048663585 |
| ca-CondMat                       | 23133 | 186936 | 267392475 |
| ca-dblp-2010                     | 226415 | 716462 | 25631272858 |
| ca-dblp-2012                     | 317082 | 1049868 | 50269606035 |
| ca-HepPh                         | 12008 | 237010 | 71865026 |
| ca-HepTh                         | 9877 | 51971 | 48730532 |
| ca-MathSciNet                    | 332689 | 820644 | 55340331061 |
| ia-email-EU                      | 32430 | 54397 | 525814268 |
| ia-reality-call                  | 6809 | 9484 | 23175161 |
| ia-retweet-pol                   | 18470 | 61157 | 170518528 |
| ia-wiki-Talk                     | 92117 | 360767 | 4242456136 |
| rt-pol                           | 18470 | 61157 | 170518528 |
| rt_barackobama                   | 9631 | 9826 | 46373070 |
| soc-epinions                     | 63947 | 606512 | 2044034866 |
| soc-gplus                        | 23628 | 39242 | 279113764 |
| tech-as-caida2007                | 26477 | 53383 | 350475620 |
| tech-internet-as                 | 40164 | 85123 | 806508407 |
| tech-pgp                         | 10680 | 24340 | 57012200 |
| tech-WHOIS                       | 7476 | 56943 | 27892083 |
| web-arabic-2005                  | 163598 | 1747269 | 13380487332 |
| web-baidu-baike-related         | 415641 | 3284387 | 86375643874 |
| web-it-2004                      | 509338 | 7178413 | 129705675378 |
| web-NotreDame                    | 325729 | 1497134 | 53048356451 |
| web-sk-2005                      | 121422 | 334419 | 7371377334 |
But the size of $E^C$ can be problematic...

| Instance                  | $|V|$ | $|E|$  | $|E^C|$ |
|---------------------------|-----|-------|--------|
| comm-n10000               | 10000 | 10000 | 49995000 |
| ca-AstroPh                | 18772 | 396160 | 175807218 |
| ca-citeseer               | 227322 | 814136 | 25836945367 |
| ca-coauthors-dblp         | 540488 | 15245731 | 146048663585 |
| ca-CondMat                | 23133 | 186936 | 267392475 |
| ca-dblp-2010              | 226415 | 716462 | 25631272858 |
| ca-dblp-2012              | 317082 | 1049868 | 50269606035 |
| ca-HepPh                  | 12008 | 237010 | 71865026 |
| ca-HepTh                  | 9877 | 51971 | 48730532 |
| ca-MathSciNet             | 332689 | 820644 | 55340331061 |
| ia-email-EU               | 32430 | 54397 | 525814268 |
| ia-reality-call           | 6809 | 9484 | 23175161 |
| ia-retweet-pol            | 18470 | 61157 | 170518528 |
| ia-wiki-Talk              | 92117 | 360767 | 4242456136 |
| rt-pol                    | 18470 | 61157 | 170518528 |
| rt_barackobama            | 9631 | 9826 | 46373070 |
| soc-epinions              | 63947 | 606512 | 2044034866 |
| soc-gplus                 | 23628 | 39242 | 279113764 |
| tech-as-caida2007         | 26477 | 53383 | 350475620 |
| tech-internet-as          | 40164 | 85123 | 806508407 |
| tech-pgp                  | 10680 | 24340 | 57012200 |
| tech-WHOIS                | 7476 | 56943 | 27892083 |
| web-arabic-2005           | 163598 | 1747269 | 13380487332 |
| web-baidu-baike-related   | 415641 | 3284387 | 86375643874 |
| web-it-2004               | 509338 | 7178413 | 129705675378 |
| web-NotreDame             | 325729 | 1497134 | 53048356451 |
| web-sk-2005               | 121422 | 334419 | 7371377334 |

$|E^C| = \frac{|E| \times (|E| - 1)}{2} - |E|$
But the size of $E^C$ can be problematic...

| Instance                  | $|V|$  | $|E|$  | $|E|^C$ |
|---------------------------|-------|-------|--------|
| comm-n10000               | 10000  | 10000  | 49995000 |
| ca-AstroPh                | 18772  | 396160 | 175807218 |
| ca-citeseer               | 227322 | 814136 | 25836945367 |
| ca-coauthors-dblp         | 540488 | 15245731 | 146048663585 |
| ca-CondMat                | 23133  | 186936 | 267392475 |
| ca-dblp-2010              | 226415 | 716462 | 25631272858 |
| ca-dblp-2012              | 317082 | 1049868 | 50269606035 |
| ca-HepPh                  | 12008  | 237010 | 71865026 |
| ca-HepTh                  | 9877   | 51971  | 48730532 |
| ca-MathSciNet             | 332689 | 820644 | 55340331061 |
| ia-email-EU               | 32430  | 54397  | 525814268 |
| ia-reality-call           | 6809   | 9484   | 23175161 |
| ia-retweet-pol            | 18470  | 61157  | 170518528 |
| ia-wiki-Talk              | 92117  | 360767 | 4242456136 |
| rt-pol                    | 18470  | 61157  | 170518528 |
| rt_barackobama            | 9631   | 9826   | 46373070 |
| soc-epinions              | 63947  | 606512 | 2044034866 |
| soc-gplus                 | 23628  | 39242  | 279113764 |
| tech-as-caida2007         | 26477  | 53383  | 350475620 |
| tech-internet-as          | 40164  | 85123  | 806508407 |
| tech-pgp                  | 10680  | 24340  | 57012200 |
| tech-WHOIS                | 7476   | 56943  | 27892083 |
| web-arabic-2005           | 163598 | 1747269 | 13380487332 |
| web-baidu-baike-related   | 415641 | 3284387 | 86375643874 |
| web-it-2004               | 509338 | 7178413 | 129705675378 |
| web-NotreDame             | 325729 | 1497134 | 53048356451 |
| web-sk-2005               | 121422 | 334419 | 7371377334 |

Unrealistic to model with SAT on sparse graphs

$$|E^C| = \frac{|E| \times (|E| - 1)}{2} - |E|$$
How to reduce the encoding size?

- Main hurdle:

  SAT-based approaches based on $G^C = (V, E^C)$ will not scale...
  And $G = (V, E)$ is much smaller than $G^C = (V, E^C)$
How to reduce the encoding size?

- Main hurdle:

SAT-based approaches based on $G^C = (V, E^C)$ will not scale...
And $G = (V, E)$ is much smaller than $G^C = (V, E^C)$

- Can we model MaxClique using solely $G$?
Another take at solving MaxClique with SAT

- Revisit the original decision problem:

  Given $G = (V, E)$, is there a clique of size $K$?
Another take at solving MaxClique with SAT

- Revisit the original decision problem:

  Given $G = (V, E)$, is there a clique of size $K$?

- First, one **must** pick exactly $K$ vertices:

  $$\sum_{u \in V} x_u = K$$
Another take at solving MaxClique with SAT

- Revisit the original decision problem:

  Given $G = (V, E)$, is there a clique of size $K$?

- First, one **must** pick exactly $K$ vertices:

  $$\sum_{u \in V} x_u = K$$

- And second, if a vertex $u \in V$ is picked (i.e. $x_u = 1$), **then** $K - 1$ of its neighbours **must** also be picked!

  $$x_u \rightarrow \left( \sum_{v \in \text{Adj}(u)} x_v = K - 1 \right)$$
Part 5

A Glimpse of the Future
What next?

• Oracle-based computing
  – Problems beyond NP: optimization, quantification, enumeration, (approximate) counting

• ...
What next?

• Oracle-based computing
  – Problems beyond NP: optimization, quantification, enumeration, (approximate) counting

• Arms race for proof systems stronger than resolution/clause learning
  – Cutting Planes (CP)
  – Extended Resolution (and equivalent)

• ...

• ...
What next?

• Oracle-based computing
  – Problems beyond NP: optimization, quantification, enumeration, (approximate) counting

• Arms race for proof systems stronger than resolution/clause learning
  – Cutting Planes (CP)
  – Extended Resolution (and equivalent)

• Verification of ML models with SAT/SMT

• ...

...
What next?

- **Oracle-based computing**
  - Problems beyond NP: optimization, quantification, enumeration, (approximate) counting

- **Arms race for proof systems stronger than resolution/clause learning**
  - Cutting Planes (CP)
  - Extended Resolution (and equivalent)

- **Verification of ML models with SAT/SMT**

- **Scalable explainable AI/ML**
  - Deep NNs operate as black-boxes
  - Often important to provide small/intuitive explanations for predictions made

- ...
Some final notes

- SAT is a low-level, but very powerful problem solving paradigm
  - PySAT suggests a way to tackle this drawback, but there are others

- There is an ongoing revolution on problem solving with SAT oracles

- The use of SAT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.
Some final notes

- SAT is a low-level, but very powerful problem solving paradigm
  - PySAT suggests a way to tackle this drawback, but there are others

- There is an ongoing revolution on problem solving with SAT oracles

- The use of SAT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.

- Many fascinating research topics out there!
  - Connections with ML seem unavoidable
Sample of tools

- PySAT
- SAT solvers:
  - MiniSat
  - Glucose
- MaxSAT solvers:
  - RC2
  - MSCG
  - OpenWBO
  - MaxHS
- MUS extractors:
  - MUSer
- MCS extractors:
  - mcsXL
  - LBX
  - MCSIs
- Many other tools available from the ReasonLab server
Questions?
References


References IV

[BBR09] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel.  
New encodings of pseudo-boolean constraints into CNF.  

Diagnosing and solving over-determined constraint satisfaction problems.  

Evaluating CDCL restart schemes.  
In *Sixth Pragmatics of SAT workshop*, 2015.

[Bie08] Armin Biere.  
PicoSAT essentials.  


References VI


References VIII

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland.
A machine program for theorem-proving.

[DP60] Martin Davis and Hilary Putnam.
A computing procedure for quantification theory.

Explanation-based generalisation of failures.

[ES03] Niklas Eén and Niklas Sörensson.
An extensible SAT-solver.

[ES06] Niklas Eén and Niklas Sörensson.
Translating pseudo-boolean constraints into SAT.
[FM06] Zhaohui Fu and Sharad Malik.  
On solving the partial MAX-SAT problem.  

Solving non-boolean satisfiability problems with stochastic local search.  

[FS02] Torsten Fahle and Meinolf Sellmann.  
Cost based filtering for the constrained knapsack problem.  

The log-support encoding of CSP into SAT.  
References


References XIII

A SAT-based approach to learn explainable decision sets.

[JHB12] Matti Järvisalo, Marijn Heule, and Armin Biere.
Inprocessing rules.

QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems.

[Kas90] Simon Kasif.
On the parallel complexity of discrete relaxation in constraint satisfaction networks.


References XV


References XVI


<table>
<thead>
<tr>
<th>Reference</th>
<th>Authors</th>
<th>Title</th>
<th>Conference/Book Details</th>
</tr>
</thead>
</table>
GRASP - a new search algorithm for satisfiability.

GRASP: A search algorithm for propositional satisfiability.

[NIPM18] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao
Marques-Silva.
Learning optimal decision trees with SAT.

A lightweight component caching scheme for satisfiability solvers.


References XX

[Rob65] John Alan Robinson.
A machine-oriented logic based on the resolution principle.

[SB09] Niklas Sörensson and Armin Biere.
Minimizing learned clauses.

[Sel03] Meinolf Sellmann.
Approximated consistency for knapsack constraints.

[Sin05] Carsten Sinz.
Towards an optimal CNF encoding of boolean cardinality constraints.
Improved design debugging using maximum satisfiability.

[SP04] Sathiamoorthy Subbarayan and Dhiraj K. Pradhan.
NiVER: Non increasing variable elimination resolution for preprocessing SAT instances.

Learning back-clauses in SAT.

[Stu13] Peter J. Stuckey.
There are no CNF problems.
[SZGN17] Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques.


[Wal00] Toby Walsh.

A linear-time transformation of linear inequalities into conjunctive normal form.
[ZM03] Lintao Zhang and Sharad Malik. 
Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications. 

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. 
Efficient conflict driven learning in boolean satisfiability solver. 

[ZS00] Hantao Zhang and Mark E. Stickel. 
Implementing the Davis-Putnam method. 