SAT Oracles, for NP-Complete Problems and Beyond

Combinatorial problem solving using SAT solvers

Daniel Le Berre

CNRS - Université d’Artois

ECSA/SPLC 19, Paris, September 11-13, 2019
Purpose of this talk

- Using SAT solvers are black boxes
- Importance of the interaction with the solver
- Importance of encodings
- When encodings are too large
SAT, SAT Oracle, SAT Solver

Importance of the interaction with the solver

Importance of the encodings

When encodings are too large
The SAT problem: textbook definition

Definition
Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: Is there an assignment of the $n$ variables that satisfies all those clauses?
The SAT problem: textbook definition

**Definition**

Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: Is there an assignment of the $n$ variables that satisfies all those clauses?

**Example**

$C_1 = \{\neg a \lor b, \neg b \lor c\} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b).(b' + c)$

$C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c$

For $C_1$, the answer is **yes**, for $C_2$ the answer is **no**

$C_1 \models \neg(a \land \neg c) = \neg a \lor c$
Definition

Input : A set of clauses $C$ built from a propositional language with $n$ variables.
Output : If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else answer UNSAT.
Definition
Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else answer UNSAT.

Example

$$C_1 = \{ \neg a \lor b, \neg b \lor c \} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c$$

For $C_1$, one answer is $\{a, b, c\}$, for $C_2$ the answer is UNSAT.
Definition

Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else answer UNSAT.

Example

$$C_1 = \{ \neg a \lor b, \neg b \lor c \} = (-a \lor b) \land (\neg b \lor c) = (a' + b).(b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c$$

For $C_1$, one answer is $\{a, b, c\}$, for $C_2$ the answer is UNSAT.

SAT answers can be checked: trusted model oracle
Definition
Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else provide a subset of $C$ which cannot be satisfied.
Definition
Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else provide a subset of $C$ which cannot be satisfied.

Example

$$C_1 = \{\neg a \lor b, \neg b \lor c\} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b) \land (b' + c)$$

$$C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c$$

For $C_1$, one answer is $\{a, b, c\}$, for $C_2$ the answer is $C_2$
The SAT problem solver: practical point of view 2/3

Definition

Input: A set of clauses $C$ built from a propositional language with $n$ variables.
Output: If there is an assignment of the $n$ variables that satisfies all those clauses, provide such assignment, else provide a subset of $C$ which cannot be satisfied.

Example

\[ C_1 = \{\neg a \lor b, \neg b \lor c\} = (\neg a \lor b) \land (\neg b \lor c) = (a' + b).(b' + c) \]

\[ C_2 = C_1 \cup \{a, \neg c\} = C_1 \land a \land \neg c \]

For $C_1$, one answer is $\{a, b, c\}$, for $C_2$ the answer is $C_2$

UNSAT core may explain inconsistency if much smaller than $C$: informative UNSAT oracle
Definition

Allow the solver to decide the satisfiability of a formula with:

- increasing number of constraints
- provided some “assumptions” are satisfied
Definition
Allow the solver to decide the satisfiability of a formula with:
- increasing number of constraints
- provided some “assumptions” are satisfied

Example

\[ C = \{ s_1 \lor \neg a \lor b, \ s_1 \lor \neg b \lor c, \ s_2 \lor a, \ s_2 \lor \neg c \} \]

\[ C_1 \equiv C \land \neg s_1 \land s_2 \]

\[ C_2 \equiv C \land \neg s_1 \land \neg s_2 \]
The SAT problem solver: practical point of view 3/3

Definition
Allow the solver to decide the satisfiability of a formula with:
▶ increasing number of constraints
▶ provided some “assumptions” are satisfied

Example

\[ C = \{s_1 \lor \neg a \lor b, s_1 \lor \neg b \lor c, s_2 \lor a, s_2 \lor \neg c\} \]

\[ C_1 \equiv C \land \neg s_1 \land s_2 \]

\[ C_2 \equiv C \land \neg s_1 \land \neg s_2 \]

The solver is considered as a stateful system: as long as the constraints are satisfiable, learn clauses can be kept: incremental SAT oracle
A short history of SAT in one slide

- **60’s First algorithms** [DP60, DLL62, Robinson65]
  \[ \text{DP} + \text{DLL} = \text{DPLL} \]

- **70’s SAT is NP-complete** [Cook71]
  \textbf{SAT is one of the simplest hard problems in CS}

- **90’s Applications, Solvers, Competitions**
  Planning as Satisfiability, Alloy, Bounded Model Checking
  Solvers available in source (GRASP, SATO, RELSAT, WALKSAT, and many more)
  Padderborn (92), DIMACS@Rutgers (93) and Beijing (96)

- **00’s Revolution, Competitions, Adoption**
  Chaff (2001) and Minisat (2003)
  Yearly competition or race
  SAT increasingly used both in academia and industry

- **10’s NP and Beyond NP**
  MAXSAT, QBF
  Largest mathematical proof (Pythagorean triples, 200TB)
In the present paper, a uniform proof procedure for quantification theory is given which is feasible for use with some rather complicated formulas and which does not ordinarily lead to exponentiation. The superiority of the present procedure over those previously available is indicated in part by the fact that a formula on which Gilmore’s routine for the IBM 704 causes the machine to compute for 21 minutes without obtaining a result was worked successfully by hand computation using the present method in 30 minutes [Davis and Putnam, 1960].

The well-formed formula (...) which was beyond the scope of Gilmore’s program was proved in under two minutes with the present program [Davis et al., 1962]
Outline

SAT, SAT Oracle, SAT Solver

Importance of the interaction with the solver

Importance of the encodings

When encodings are too large
How to solve \textbf{MaxSat MinUnsat} with SAT?

- Associate to each clause a weight (penalty) \( w_i \) taken into account if the clause is violated: \textit{Soft clauses} \( S \).
- Special weight \((\infty)\) for clauses that cannot be violated: \textit{hard clauses} \( H \).

\textbf{Definition (Partial Weighted MaxSat)}

Find a model \( M \) of \( H \) that minimizes \( \text{weight}(M, S) \) such that:

- \( \text{weight}(M, (c_i, w_i)) = 0 \) if \( M \) satisfies \( c_i \), else \( w_i \).
- \( \text{weight}(M, S) = \sum_{wc \in S} \text{weight}(M, wc) \)

Simply called \textbf{MaxSAT} if \( k = 1 \) and \( H = \emptyset \).
How to solve MaxSat MinUnsat with SAT?

- Associate to each clause a weight (penalty) $w_i$ taken into account if the clause is violated: **Soft clauses** $S$.
  $\left( \neg a \lor b, 6 \right) \land \left( \neg b \lor c, 8 \right)$

- Special weight ($\infty$) for clauses that cannot be violated: **hard clauses** $H$

**Definition (Partial Weighted MaxSat)**

Find a model $M$ of $H$ that minimizes $weight(M, S)$ such that:

- $weight(M, (c_i, w_i)) = 0$ if $M$ satisfies $c_i$, else $w_i$.
- $weight(M, S) = \sum_{wc \in S} weight(M, wc)$

Simply called MaxSAT if $k = 1$ and $H = \emptyset$
How to solve \( \text{MaxSat MinUnsat} \) with SAT?

- Associate to each clause a weight (penalty) \( w_i \) taken into account if the clause is violated: \( \text{Soft clauses } S \).
  \[
  (\neg a \lor b, 6) \land (\neg b \lor c, 8)
  \]

- Special weight \( \infty \) for clauses that cannot be violated: \( \text{hard clauses } H \)
  \[
  (a, \infty) \land (\neg c, \infty)
  \]

**Definition (Partial Weighted MaxSat)**

Find a model \( M \) of \( H \) that minimizes \( \text{weight}(M, S) \) such that:

- \( \text{weight}(M, (c_i, w_i)) = 0 \) if \( M \) satisfies \( c_i \), else \( w_i \).
- \( \text{weight}(M, S) = \sum_{wc \in S} \text{weight}(M, wc) \)

Simply called MaxSAT if \( k = 1 \) and \( H = \emptyset \)
How to solve \textbf{MaxSat MinUnsat} with SAT?

- Associate to each clause a weight (penalty) \( w_i \) taken into account if the clause is violated: \textbf{Soft clauses} \( S \).
  \( (\neg a \lor b, 6) \land (\neg b \lor c, 8) \)

- Special weight \( (\infty) \) for clauses that cannot be violated: \textbf{hard clauses} \( H \)
  \( (a, \infty) \land (\neg c, \infty) \)

**Definition (Partial Weighted MaxSat)**

Find a model \( M \) of \( H \) that minimizes \( weight(M, S) \) such that:

- \( weight(M, (c_i, w_i)) = 0 \) if \( M \) satisfies \( c_i \), else \( w_i \).
- \( weight(M, S) = \sum_{wc \in S} weight(M, wc) \) weight of \( \{a, \neg b, \neg c\} \) is 6

Simply called \textbf{MaxSAT} if \( k = 1 \) and \( H = \emptyset \)
Linear Search for solving MaxSAT

<table>
<thead>
<tr>
<th>x6, x2</th>
<th>¬x6, x2</th>
<th>¬x2, x1</th>
<th>¬x1</th>
</tr>
</thead>
<tbody>
<tr>
<td>¬x6, x8</td>
<td>x6, ¬x8</td>
<td>x2, x4</td>
<td>¬x4, x5</td>
</tr>
<tr>
<td>x7, x5</td>
<td>¬x7, x5</td>
<td>¬x5, x3</td>
<td>¬x3</td>
</tr>
</tbody>
</table>

Example CNF formula ($k = 1$ for each clause, not displayed)
Linear Search for solving MaxSAT

\[
\begin{align*}
&x_6, x_2, b_7 & \neg x_6, x_2, b_8 & \neg x_2, x_1, b_1 & \neg x_1, b_2 \\
&\neg x_6, x_8, b_9 & x_6, \neg x_8, b_{10} & x_2, x_4, b_3 & \neg x_4, x_5, b_4 \\
&x_7, x_5, b_{11} & \neg x_7, x_5, b_{12} & \neg x_5, x_3, b_5 & \neg x_3, b_6
\end{align*}
\]

Add selector or **blocking** variables \( b_i \);
Linear Search for solving MaxSAT

<table>
<thead>
<tr>
<th>$x_6, x_2, b_7$</th>
<th>$\neg x_6, x_2, b_8$</th>
<th>$\neg x_2, x_1, b_1$</th>
<th>$\neg x_1, b_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\neg x_6, x_8, b_9$</td>
<td>$x_6, \neg x_8, b_{10}$</td>
<td>$x_2, x_4, b_3$</td>
<td>$\neg x_4, x_5, b_4$</td>
</tr>
<tr>
<td>$x_7, x_5, b_{11}$</td>
<td>$\neg x_7, x_5, b_{12}$</td>
<td>$\neg x_5, x_3, b_5$</td>
<td>$\neg x_3, b_6$</td>
</tr>
</tbody>
</table>

Formula is SAT; eg model M contains

$b_1, \neg b_2, b_3, \neg b_4, b_5, \neg b_7, \neg b_8, \neg b_9, b_{10}, \neg b_{11}, b_{12}$
Linear Search for solving MaxSAT

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8, b_9 \quad x_6, \neg x_8, b_{10} \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

\[ \sum_{i=1}^{12} b_i < 5 \]

Bound the number of constraints to be relaxed: \(|M \cap B| = 5\)
Linear Search for solving MaxSAT

\[
\begin{align*}
\text{x}_6, \text{x}_2, b_7 & \quad & \neg \text{x}_6, \text{x}_2, b_8 & \quad & \neg \text{x}_2, \text{x}_1, b_1 & \quad & \neg \text{x}_1, b_2 \\
\neg \text{x}_6, \text{x}_8, b_9 & \quad & \text{x}_6, \neg \text{x}_8, b_{10} & \quad & \text{x}_2, \text{x}_4, b_3 & \quad & \neg \text{x}_4, \text{x}_5, b_4 \\
\text{x}_7, \text{x}_5, b_{11} & \quad & \neg \text{x}_7, \text{x}_5, b_{12} & \quad & \neg \text{x}_5, \text{x}_3, b_5 & \quad & \neg \text{x}_3, b_6 \\
\sum_{i=1}^{12} b_i < 5
\end{align*}
\]

Formula is (again) SAT; eg model contains
\( b_1, \neg b_2, \neg b_3, \neg b_4, \neg b_5, \neg b_7, \neg b_8, \neg b_9, \neg b_{10}, \neg b_{11}, b_{12} \)
Linear Search for solving MaxSAT

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8, b_9 \quad x_6, \neg x_8, b_{10} \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

\[ \sum_{i=1}^{12} b_i < 2 \]

Bound the number of constraints to be relaxed \(|M \cap B| = 2| \]
Linear Search for solving MaxSAT

\[
\begin{align*}
& x_6, x_2, b_7 & -x_6, x_2, b_8 & -x_2, x_1, b_1 & -x_1, b_2 \\
& -x_6, x_8, b_9 & x_6, -x_8, b_{10} & x_2, x_4, b_3 & -x_4, x_5, b_4 \\
& x_7, x_5, b_{11} & -x_7, x_5, b_{12} & -x_5, x_3, b_5 & -x_3, b_6 \\
\sum_{i=1}^{12} b_i & < 2
\end{align*}
\]

Instance is now UNSAT
Linear Search for solving MaxSAT

MaxSAT solution is $|\varphi| - |M \cap B| = 12 - 2 = 10$
Note that ...

- No initial upper or lower bounds: the first model provides a first upper bound.
- In practice, the objective function can be used to guide the search.
- The procedure follows a SAT, SAT, SAT, SAT, ..., UNSAT pattern with linear search.
- Binary search is possible but:
  - SAT answer is usually faster than UNSAT
  - the solver must be reset in case on unsatisfiability.
- In lucky case, two calls to the SAT solver are sufficient (one SAT + one UNSAT).
- Used in Sat4j since 2006, was state-of-the-art in 2009.
- Main issue: how to represent the bound constraint?
From Unsat Core computation to MaxSat: MSU

Other SAT-based approaches in practical Max Sat solving rely on unsat core computation [Fu and Malik 2006]:

- Compute one unsat core $C'$ of the formula $C$
- Relax it by replacing $C'$ by $\{ r_i \lor C_i | C_i \in C' \}$
- Add the constraint $\sum r_i \leq 1$ to $C$
- Repeat until the formula is satisfiable
- If $MinUnsat(C) = k$, requires $k + 1$ loops.

Many improvement since then (PM1, PM2, MsUncore, etc): works for Weighted Max Sat, reduction of the number of relaxation variables, etc.
Fu&Malik’s Algorithm: msu1.0

Example CNF formula
Formula is **UNSAT**; Get unsat core
Fu&Malik’s Algorithm: msu1.0

\[ x_6, x_2 \quad \neg x_6, x_2 \quad \neg x_2, x_1, b_1 \quad \neg x_1, b_2 \]

\[ \neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5 \quad \neg x_7, x_5 \quad \neg x_5, x_3, b_5 \quad \neg x_3, b_6 \]

\[ \sum_{i=1}^{6} b_i \leq 1 \]

Add blocking variables and AtMost1 constraint
Fu&Malik’s Algorithm: msu1.0

\[
\begin{align*}
&x_6, x_2 & \neg x_6, x_2 & \neg x_2, x_1, b_1 & \neg x_1, b_2 \\
&\neg x_6, x_8 & x_6, \neg x_8 & x_2, x_4, b_3 & \neg x_4, x_5, b_4 \\
&x_7, x_5 & \neg x_7, x_5 & \neg x_5, x_3, b_5 & \neg x_3, b_6 \\
\sum_{i=1}^{6} b_i & \leq 1
\end{align*}
\]

Formula is (again) **UNSAT**; Get unsat core
Fu&Malik’s Algorithm: msu1.0

\[ x_6, x_2, b_7 \quad \neg x_6, x_2, b_8 \quad \neg x_2, x_1, b_1, b_9 \quad \neg x_1, b_2, b_{10} \]

\[ \neg x_6, x_8 \quad x_6, \neg x_8 \quad x_2, x_4, b_3 \quad \neg x_4, x_5, b_4 \]

\[ x_7, x_5, b_{11} \quad \neg x_7, x_5, b_{12} \quad \neg x_5, x_3, b_5, b_{13} \quad \neg x_3, b_6, b_{14} \]

\[ \sum_{i=1}^{6} b_i \leq 1 \quad \sum_{i=7}^{14} b_i \leq 1 \]

Add new **blocking variables** and **AtMost1 constraint**
Fu&Malik’s Algorithm: msu1.0

\[
\begin{align*}
  x_6, x_2, b_7 & \quad \neg x_6, x_2, b_8 & \quad \neg x_2, x_1, b_1, b_9 & \quad \neg x_1, b_2, b_{10} \\
  \neg x_6, x_8 & \quad x_6, \neg x_8 & \quad x_2, x_4, b_3 & \quad \neg x_4, x_5, b_4 \\
  x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} & \quad \neg x_5, x_3, b_5, b_{13} & \quad \neg x_3, b_6, b_{14} \\
  \sum_{i=1}^{6} b_i \leq 1 & \quad \sum_{i=7}^{14} b_i \leq 1
\end{align*}
\]

Instance is now SAT
### Fu&Malik’s Algorithm: msu1.0

<table>
<thead>
<tr>
<th>x_{6}, x_{2}, b_{7}</th>
<th>\neg x_{6}, x_{2}, b_{8}</th>
<th>\neg x_{2}, x_{1}, b_{1}, b_{9}</th>
<th>\neg x_{1}, b_{2}, b_{10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\neg x_{6}, x_{8}</td>
<td>x_{6}, \neg x_{8}</td>
<td>x_{2}, x_{4}, b_{3}</td>
<td>\neg x_{4}, x_{5}, b_{4}</td>
</tr>
<tr>
<td>x_{7}, x_{5}, b_{11}</td>
<td>\neg x_{7}, x_{5}, b_{12}</td>
<td>\neg x_{5}, x_{3}, b_{5}, b_{13}</td>
<td>\neg x_{3}, b_{6}, b_{14}</td>
</tr>
</tbody>
</table>

\[
\sum_{i=1}^{6} b_{i} \leq 1 \quad \sum_{i=7}^{14} b_{i} \leq 1
\]

MaxSAT solution is $|\varphi| - I = 12 - 2 = 10$
Note that ...

- Unsat core may not be minimal
- Nice property: if \( k \) constraints must be relaxed, then the procedure requires exactly \( k + 1 \) calls to the SAT solver.
- How to represent the cardinality constraints?
Core guided MAXSAT solver can be seen as a two step procedure:

- Discover UNSAT cores of the formula
- Stop as soon as one minimal Hitting Set of the cores satisfies the formula

- The size of the HS provides the number of constraints to relax
- May require to enumerate all MUS of a formula
- Or less if lucky
MaxHS principle

\[
\begin{align*}
\neg x_6, x_2, b_7 & \quad \neg \neg x_6, x_2, b_8 & \quad \neg \neg x_2, x_1, b_1 & \quad \neg \neg x_1, b_2 \\
\neg \neg x_6, x_8, b_9 & \quad x_6, \neg x_8, b_{10} & \quad x_2, x_4, b_3 & \quad \neg x_4, x_5, b_4 \\
x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} & \quad \neg x_5, x_3, b_5 & \quad \neg x_3, b_6
\end{align*}
\]

Cores = \{\} \quad HS = \emptyset
MaxHS principle

\[
\begin{align*}
  x_6, x_2, b_7 & \quad \neg x_6, x_2, b_8 \\
  \neg x_6, x_8, b_9 & \quad x_6, \neg x_8, b_{10} \\
  x_7, x_5, b_{11} & \quad \neg x_7, x_5, b_{12} \\
  \neg x_1, b_2 & \\
  \neg x_4, x_5, b_4 & \\
  \neg x_3, b_6 &
\end{align*}
\]

\[
\{ b_1, b_2, b_3, b_4, b_5, b_6 \}\]

\[
HS = \{ b_4 \}\]
MaxHS principle

\[
\begin{array}{cccc}
 x_6, x_2, b_7 & \neg x_6, x_2, b_8 & \neg x_2, x_1, b_1 & \neg x_1, b_2 \\
 \neg x_6, x_8, b_9 & x_6, \neg x_8, b_{10} & x_2, x_4, b_3 & \neg x_4, x_5, b_4 \\
 x_7, x_5, b_{11} & \neg x_7, x_5, b_{12} & \neg x_5, x_3, b_5 & \neg x_3, b_6 \\
\end{array}
\]

\[
\{ \{ b_1, b_2, b_3, b_4, b_5, b_6 \}, \{ b_1, b_2, b_7, b_8 \} \} \quad \text{HS} = \{ b_1 \} 
\]
MaxHS principle

\[
\begin{align*}
& x_6, x_2, b_7 & \quad & \neg x_6, x_2, b_8 & \quad & \neg x_2, x_1, b_1 & \quad & \neg x_1, b_2 \\
& \neg x_6, x_8, b_9 & \quad & x_6, \neg x_8, b_{10} & \quad & x_2, x_4, b_3 & \quad & \neg x_4, x_5, b_4 \\
& x_7, x_5, b_{11} & \quad & \neg x_7, x_5, b_{12} & \quad & \neg x_5, x_3, b_5 & \quad & \neg x_3, b_6
\end{align*}
\]

\[
\{ \{ b_1, b_2, b_3, b_4, b_5, b_6 \}, \{ b_1, b_2, b_7, b_8 \}, \{ b_{11}, b_{12}, b_5, b_6 \} \} \\
HS = \{ b_2, b_5 \}
\]
Instance is SAT. MaxSAT solution is $12 - |\{b_2, b_5\}| = 10$
3 ways to solve the same [optimization] problem

- Take advantage of SAT solvers feedback: model or core
- No single approach outperforms the others
- Core-guided and MaxHS work best currently on "application" benchmarks (not crafted ones)

Linear Search or Core-Guided approaches require encoding cardinality constraints in CNF (or use native support for such constraints as found in Sat4j)
Outline

SAT, SAT Oracle, SAT Solver

Importance of the interaction with the solver

Importance of the encodings

When encodings are too large
Quick question for the audience

How would you encode

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \leq 1 \]

as a CNF?
Quick question for the audience

How would you encode

\[ x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} \leq 1 \]

as a CNF?

\[ \neg x_1 \lor \neg x_2, \neg x_1 \lor \neg x_3, \neg x_1 \lor \neg x_4, \neg x_1 \lor \neg x_5, \neg x_1 \lor \neg x_6, \]
\[ \neg x_1 \lor \neg x_7, \neg x_1 \lor \neg x_8, \neg x_1 \lor \neg x_9, \neg x_1 \lor \neg x_{10}, \]
\[ \neg x_2 \lor \neg x_3, \neg x_2 \lor \neg x_4, \neg x_2 \lor \neg x_5, \neg x_2 \lor \neg x_6, \]
\[ \neg x_2 \lor \neg x_7, \neg x_2 \lor \neg x_8, \neg x_2 \lor \neg x_9, \neg x_2 \lor \neg x_{10}, \]
\[ \neg x_3 \lor \neg x_4, \neg x_3 \lor \neg x_5, \neg x_3 \lor \neg x_6, \neg x_3 \lor \neg x_7, \neg x_3 \lor \neg x_8, \]
\[ \neg x_3 \lor \neg x_9, \neg x_3 \lor \neg x_{10}, \]
\[ \neg x_4 \lor \neg x_5, \neg x_4 \lor \neg x_6, \neg x_4 \lor \neg x_7, \neg x_4 \lor \neg x_8, \neg x_4 \lor \neg x_9, \neg x_4 \lor \neg x_{10} \]
\[ \neg x_5 \lor \neg x_6, \neg x_5 \lor \neg x_7, \neg x_5 \lor \neg x_8, \neg x_5 \lor \neg x_9, \neg x_5 \lor \neg x_{10}, \]
\[ \neg x_6 \lor \neg x_7, \neg x_6 \lor \neg x_8, \neg x_6 \lor \neg x_9, \neg x_6 \lor \neg x_{10}, \]
\[ \neg x_7 \lor \neg x_8, \neg x_7 \lor \neg x_9, \neg x_7 \lor \neg x_{10}, \]
\[ \neg x_8 \lor \neg x_9, \neg x_8 \lor \neg x_{10}, \neg x_8 \lor \neg x_{10}. \]

Pairwise encoding, 45 binary clauses
Some known encodings for cardinality constraints

Short list of known encodings:

- **Pairwise encoding** [Cook et al., 1987]
- Nested encoding
- **Two product encoding** [Chen, 2010]
- Sequential encoding [Sinz, 2005]
- Commander encoding [Frisch and Giannaros, 2010]
- Ladder encoding [Gent and Nightingale, 2004]
- Adder encoding [Eén and Sörensson, 2006]
- Cardinality Networks [Asín et al., 2009]
- ...

...
Two product encoding

$$\sum_{i=1}^{10} x_i \leq 1$$
Two product encoding


Selection clauses

\[
\begin{align*}
(\neg x_1 & \lor r_1) \\
(\neg x_1 & \lor c_1) \\
(\neg x_2 & \lor c_2) \\
(\neg x_3 & \lor r_1) \\
(\neg x_3 & \lor c_3) \\
(\neg x_4 & \lor r_1) \\
(\neg x_4 & \lor c_4) \\
(\neg x_5 & \lor c_1) \\
(\neg x_6 & \lor r_2) \\
(\neg x_6 & \lor c_2) \\
(\neg x_7 & \lor r_2) \\
(\neg x_7 & \lor c_3) \\
(\neg x_8 & \lor c_4) \\
(\neg x_9 & \lor r_3) \\
(\neg x_9 & \lor c_1) \\
(\neg x_{10} & \lor r_3) \\
(\neg x_{10} & \lor c_2)
\end{align*}
\]

final AtMost-1

\[
\begin{align*}
(\neg r_1 & \lor \neg r_2) \\
(\neg r_1 & \lor \neg r_3) \\
(\neg r_2 & \lor \neg r_3) \\
(\neg c_1 & \lor \neg c_2) \\
(\neg c_1 & \lor \neg c_3) \\
(\neg c_1 & \lor \neg c_4) \\
(\neg c_2 & \lor \neg c_3) \\
(\neg c_2 & \lor \neg c_4) \\
(\neg c_3 & \lor \neg c_4)
\end{align*}
\]

encoding \[ \sum_{i=1}^{10} x_i \leq 1 \]
Two product encoding

Selection clauses

(¬x₁ ∨ r₁)  (¬x₁ ∨ c₁)  (¬x₂ ∨ r₁)
(¬x₂ ∨ c₂)  (¬x₃ ∨ r₁)  (¬x₃ ∨ c₃)
(¬x₄ ∨ r₁)  (¬x₄ ∨ c₄)  (¬x₅ ∨ r₂)
(¬x₅ ∨ c₁)  (¬x₆ ∨ r₂)  (¬x₆ ∨ c₂)
(¬x₇ ∨ r₂)  (¬x₇ ∨ c₃)  (¬x₈ ∨ r₂)
(¬x₈ ∨ c₄)  (¬x₉ ∨ r₃)  (¬x₉ ∨ c₁)
(¬x₁₀ ∨ r₃) (¬x₁₀ ∨ c₂)

final AtMost-1

(¬r₁ ∨ ¬r₂)  (¬r₁ ∨ ¬r₃)  (¬r₂ ∨ ¬r₃)
(¬c₁ ∨ ¬c₂)  (¬c₁ ∨ ¬c₃)  (¬c₁ ∨ ¬c₄)
(¬c₂ ∨ ¬c₃)  (¬c₂ ∨ ¬c₄)  (¬c₃ ∨ ¬c₄)

encoding $\sum_{i=1}^{10} x_i \leq 1$
Two product encoding

Selection clauses

\[
\begin{align*}
(\neg x_1 \lor r_1) & \quad (\neg x_1 \lor c_1) & \quad (\neg x_2 \lor r_1) \\
(\neg x_2 \lor c_2) & \quad (\neg x_3 \lor r_1) & \quad (\neg x_3 \lor c_3) \\
(\neg x_4 \lor r_1) & \quad (\neg x_4 \lor c_4) & \quad (\neg x_5 \lor r_2) \\
(\neg x_5 \lor c_1) & \quad (\neg x_6 \lor r_2) & \quad (\neg x_6 \lor c_2) \\
(\neg x_7 \lor r_2) & \quad (\neg x_7 \lor c_3) & \quad (\neg x_8 \lor r_2) \\
(\neg x_8 \lor c_4) & \quad (\neg x_9 \lor r_3) & \quad (\neg x_9 \lor c_1) \\
(\neg x_{10} \lor r_3) & \quad (\neg x_{10} \lor c_2) \\
\end{align*}
\]

Final AtMost-1

\[
\begin{align*}
(\neg r_1 \lor \neg r_2) & \quad (\neg r_1 \lor \neg r_3) & \quad (\neg r_2 \lor \neg r_3) \\
(\neg c_1 \lor \neg c_2) & \quad (\neg c_1 \lor \neg c_3) & \quad (\neg c_1 \lor \neg c_4) \\
(\neg c_2 \lor \neg c_3) & \quad (\neg c_2 \lor \neg c_4) & \quad (\neg c_3 \lor \neg c_4) \\
\end{align*}
\]

encoding \[ \sum_{i=1}^{10} x_i \leq 1 \]
Two product encoding

\[
\text{encoding } \sum_{i=1}^{10} x_i \leq 1
\]
Two product encoding


Selection clauses

\[
(\neg x_1 \lor r_1) \quad (\neg x_1 \lor c_1) \quad (\neg x_2 \lor r_1) \\
(\neg x_2 \lor c_2) \quad (\neg x_3 \lor r_1) \quad (\neg x_3 \lor c_3) \\
(\neg x_4 \lor r_1) \quad (\neg x_4 \lor c_4) \quad (\neg x_5 \lor r_2) \\
(\neg x_5 \lor c_1) \quad (\neg x_6 \lor r_2) \quad (\neg x_6 \lor c_2) \\
(\neg x_7 \lor r_2) \quad (\neg x_7 \lor c_3) \quad (\neg x_8 \lor r_2) \\
(\neg x_8 \lor c_4) \quad (\neg x_9 \lor r_3) \quad (\neg x_9 \lor c_1) \\
(\neg x_{10} \lor r_3) \quad (\neg x_{10} \lor c_2)
\]

final AtMost-1

\[
(\neg r_1 \lor \neg r_2) \quad (\neg r_1 \lor \neg r_3) \quad (\neg r_2 \lor \neg r_3) \\
(\neg c_1 \lor \neg c_2) \quad (\neg c_1 \lor \neg c_3) \quad (\neg c_1 \lor \neg c_4) \\
(\neg c_2 \lor \neg c_3) \quad (\neg c_2 \lor \neg c_4) \quad (\neg c_3 \lor \neg c_4)
\]

encoding \[ \sum_{i=1}^{10} x_i \leq 1 \] 29 binary clauses
Cardinality/Pseudo-Boolean constraints in CNF

- Translation in CNF without adding new variables often not an option
- Various encodings available, with different properties (number of additional variables, number of generated clauses, size of generated clauses, preserve or not arc consistency, etc.)
- Different solvers may behave differently on different encodings (e.g. because of specific management of binary clauses).
- For a survey of the effect of various encodings for MaxSat, see [Martins et al., 2012].
Cardinality/Pseudo-Boolean constraints ad hoc

Other option: do not encode! (our approach in Sat4j)

- Space efficient
- Can use extended reasoning: e.g. Generalized Resolution [Hooker, 1988]
- Cannot reuse off-the-shelf solver
- Requires to maintain the constraints in the solver

When the input is in CNF, retrieve cardinality constraints [Biere et al., 2014].
Outline

SAT, SAT Oracle, SAT Solver

Importance of the interaction with the solver

Importance of the encodings

When encodings are too large
Sometimes the CNF encoding is just too large

- It is often the case that CNF encodings reach GB of space
- A popular technique in that case is to provide only parts of the constraints to the solver
- If the set of constraints is UNSAT, the original problem is UNSAT
- If the set of constraints is SAT, the model is checked against the original problem
- If the model is a solution of the original problem, the problem is solved (Lucky Outcome)
- Else new constraints (clauses) are added to prevent such kind of spurious solution (Refinement)

Counter Example Guided Abstraction Refinement
CEGAR using under-abstractions

Example

Hamiltonian cycle problem
CEGAR using over-abstractions

CEGAR-over

cegar(\(\phi\)) \rightarrow \psi \leftarrow \hat{\phi} \rightarrow \text{check}(\psi) \rightarrow \text{unsat} \rightarrow \psi \equiv_{\text{sat}} \phi \rightarrow \text{yes} \rightarrow \text{UNSAT}

\begin{align*}
\text{sat} & \quad \text{unsat} & \quad \text{unk.}
\end{align*}

Example

Planning problem, by increasing step by step the horizon; Bounded Model Checking
CounterExample Guided Abstraction Refinement

Advantages
- If problem mainly satisfiable: CEGAR-over
- If problem mainly unsatisfiable: CEGAR-under
- When check improves, CEGAR improves
- Many applications already use CEGAR

Drawbacks
- Not efficient when 50/50 chances of being SAT/UNSAT
- Not efficient when we need many refinement steps
Recursive Explore and Check Abstraction Refinement

RECAR

\[ \text{recar}(\phi) \rightarrow \psi \leftarrow \hat{\phi} \]

\[ \psi \equiv \text{sat} \phi \]

\[ \text{check}(\psi) \]

\[ \psi \leftarrow \text{refine}(\psi) \]

\[ \text{RC}(\phi, \hat{\phi}) \]

SAT

UNSAT

yes

unsat

no

sat

desat

unk.
Recursive Explore and Check Abstraction Refinement

RECAR [Lagniez et al., 2017]
- 2 levels of abstractions
  - One at the Oracle level (check($\psi$))
  - One at the Domain level (recursive call)
- Efficient even when 50/50 chance of being SAT/UNSAT
- When check improves, RECAR improves
- The return of the recursive call can reduce the number of refinements
- SAT and UNSAT shortcuts can be inverted if needed
- Totally generic, can change SAT solver by QBF/SMT/FO solver
Modal Logic $K$ is **PSPACE**-complete
[Ladner, 1977, Halpern, 1995]

What is Modal Logic $K$?

How we over-approximate a formula $\phi$?

How we under-approximate a formula $\phi$?

Is it competitive against a CEGAR approach?

Is it competitive against the state-of-the-art approaches?
Modal Logic

Modal Logic = Propositional Logic + □ and ◊

> □φ means φ is necessarily true
> ◊φ means φ is possibly true

◊φ ↔ ¬□¬φ
□φ ↔ ¬◊¬φ
Satisfiability of Modal Logic formulas

\[ \begin{align*}
\checkmark \quad \phi_1 &= \square (\bullet) \\
\times \quad \phi_2 &= \square \Diamond (\bullet)
\end{align*} \]

\[ \begin{align*}
\checkmark \quad \phi_3 &= \Diamond (\bullet \land \Diamond \neg \bullet) \\
\checkmark \quad \phi_4 &= (\bullet \lor \bullet \lor \bullet) \\
\times \quad \phi_5 &= \Diamond \Diamond (\bullet \land \square \neg \bullet)
\end{align*} \]

Figure: Example $\mathcal{K}$
Satisfiability of Modal Logic formulas

✓ $\phi_1 = \Box(\bullet)$

✗ $\phi_2 = \Box\Diamond(\bullet)$

✓ $\phi_3 = \Diamond(\bullet \land \Diamond \neg \bullet)$

✓ $\phi_4 = (\bullet \lor \bullet \lor \bullet)$

✓ $\phi_5 = \Diamond\Diamond(\bullet \land \Box \neg \bullet)$

Figure: Example $\mathcal{K}$
Suppose we want to solve the formula below, with $\chi$ huge but satisfiable...

Worst case for CEGAR using an over approximation, i.e. unrolling the Kripke structure
MoSaiC: Under-Approximation (modal logic level)

Modern SAT solvers returns ‘the reason’ why a formula with $n$ worlds is unsatisfiable ($core = \{s_1, s_2\}$)
MoSaiC: Under-Approximation (modal logic level)

We want to cut what is not part of the ‘unsatisfiability’ ($s; \notin \text{core}$)

We just create $\hat{\phi}$ smaller than $\phi$ and easier to solve. The function $RC$ from RECAR just says here: did we cut something?
MoSaiC: RECAR for Modal Logic K
MoSaiC: RECAR for Modal Logic K
Explanation of the Cactus-Plot

The graph illustrates the execution time in seconds for different algorithms over the number of instances solved. The algorithms compared are Spartacus and RECAR, with time categories for generating and SAT time.
Some tweaks improve the results
Conclusion

- SAT-based problem solving similar to assembly language programming
  - limited expressiveness
  - highly efficient
  - not for casual programmers
Conclusion

- SAT-based problem solving similar to assembly language programming
  - limited expressiveness
  - highly efficient
  - not for casual programmers

- Practical SAT solving is about
  - Encoding efficiently problems into CNF
  - Designing innovative SAT-based algorithms
  - Improving SAT solvers
  - Trusting solvers as efficient search space explorators
  - Being optimistic (versus worst case complexity)
Conclusion

- SAT-based problem solving similar to assembly language programming
  - limited expressiveness
  - highly efficient
  - not for casual programmers

- Practical SAT solving is about
  - Encoding efficiently problems into CNF
  - Designing innovative SAT-based algorithms
  - Improving SAT solvers
  - Trusting solvers as efficient search space explorators
  - Being optimistic (versus worst case complexity)

Definition (SAT-based problem solving)

- if proposal works, done
- else, try again, changing something
  (approach, encoding, solver, computers)
  driven by cause of failure
Thanks for your attention

Questions?

Bibliography II


