Maximum Satisfiability

For today’s lecture we use the slides of Matti Jaarvisalo presented at 2016 SAT Summer School

Link:
http://ssa-school-2016.it.uu.se/programme/#maxSAT
MaxSAT: Maximum Satisfiability

Matti Järvisalo

University of Helsinki

June 22, 2016
SAT-SMT-AR Summer School, Lisbon Portugal
Overview

Maximum Satisfiability—MaxSAT

Exact Boolean optimization paradigm

- Builds on the success story of Boolean satisfiability (SAT) solving
- Great recent improvements in practical solver technology
- Expanding range of real-world applications

Offers an alternative e.g. integer programming

- Solvers provide provably optimal solutions
- Propositional logic as the underlying declarative language: especially suited for inherently “very Boolean” optimization problems
Outline

Motivation

- Need for exact optimization

Basic concepts

- \textit{MaxSAT}
- Complexity
- Use in practice

Overview of algorithmic approaches to \textit{MaxSAT}

- Branch and bound
- \textit{MaxSAT} by integer programming (IP)
- SAT-based: iterative, core-guided
- SAT-IP hybrids: Implicit hitting set approach

Use of SAT solvers for \textit{MaxSAT}
Optimization

Most real-world problems involve an optimization component

Examples:
- Find a **shortest** path/plan/execution/... to a goal state
  - Planning, model checking, ...
- Find a **smallest** explanation
  - Debugging, configuration, ...
- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...
- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

High demand for automated approaches to finding good solutions to computationally hard optimization problems
Optimization

Most real-world problems involve an optimization component

Examples:

- Find a **shortest** path/plan/execution/... to a goal state
  - Planning, model checking, ...

- Find a **smallest** explanation
  - Debugging, configuration, ...

- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...

- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

High demand for automated approaches to finding good solutions to computationally hard optimization problems
Importance of Exact Optimization

Giving Up?
“The problem is NP-hard, so let’s develop heuristics/approximation algorithms.”

No!
Benefits of provably optimal solutions:
- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

Key Challenge: Scalability
Exactly solving instances of \( NP\text{-hard} \) optimization problems
Importance of Exact Optimization

Giving Up?
“The problem is NP-hard, so let’s develop heuristics / approximation algorithms.”

No!
Benefits of provably optimal solutions:
- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

Key Challenge: Scalability
Exactly solving instances of \textit{NP-hard} optimization problems
Constrained Optimization Paradigms

Mixed Integer-Linear Programming

Constraint language:
Conjunctions of linear inequalities \( \sum_{i=1}^{k} c_i x_i \leq b \)

Algorithms: e.g. Branch-and-cut w/Simplex

Normal form: integer domain variables \( x_i \), constants \( c_i, a_{ij}, b_j \)

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{k} c_i x_i \\
\text{subject to} & \quad \sum_{i=1}^{k} a_{ij} x_i \leq b_1 \\
& \quad \ldots \\
& \quad \sum_{i=1}^{k} a_{im} x_i \leq b_m
\end{align*}
\]
Constrained Optimization Paradigms

Finite-domain Constraint Optimization (COP)
- Constraint language:
  Conjunctions of high-level (global) finite-domain constraints
- Algorithms:
  Depth-first backtracking search, specialized filtering algorithms

Maximum satisfiability (MaxSAT)
- Constraint language:
  Weighted Boolean combinations of binary variables
- Algorithms:
  Building on state-of-the-art CDCL SAT solvers
  - Learning from conflicts, conflict-driven search
  - Incremental API, providing explanations for unsatisfiability
Constrained Optimization Paradigms

### Finite-domain Constraint Optimization (COP)
- **Constraint language:**
  Conjunctions of high-level (global) finite-domain constraints
- **Algorithms:**
  Depth-first backtracking search, specialized filtering algorithms

### Maximum Satisfiability (MaxSAT)
- **Constraint language:**
  Weighted Boolean combinations of binary variables
- **Algorithms:**
  Building on state-of-the-art CDCL SAT solvers
  - Learning from conflicts, conflict-driven search
  - Incremental API, providing explanations for unsatisfiability
MaxSAT Applications

- probabilistic inference
- design debugging

- maximum quartet consistency
- software package management

Max-Clique
- fault localization
- restoring CSP consistency
- reasoning over bionetworks
- MCS enumeration
- heuristics for cost-optimal planning
- optimal covering arrays
- correlation clustering
- treewidth computation
- Bayesian network structure learning
- causal discovery
- visualization
- model-based diagnosis
- cutting planes for IPs
- argumentation dynamics

[Park, 2002]
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
[Morgado and Marques-Silva, 2010]
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[Ignatiev, Janota, and Marques-Silva, 2014]
[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
[Lynce and Marques-Silva, 2011]
[Guerra and Lynce, 2012]
[Morgado, Lifiton, and Marques-Silva, 2012]
[Zhang and Bacchus, 2012]
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016a]
[Berg and Järvisalo, 2014]
[Berg, Järvisalo, and Malone, 2014]
[Hyttinen, Eberhardt, and Järvisalo, 2014]
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
[Saikko, Malone, and Järvisalo, 2015]
[Wallner, Niskanen, and Järvisalo, 2016]
MaxSAT Applications

Central to the increasing success:
Advances in MaxSAT solver technology

probabilistic inference
design debugging

maximum quartet consistency
software package management

Max-Clique
fault localization
restoring CSP consistency
reasoning over bionetworks
MCS enumeration
heuristics for cost-optimal planning
optimal covering arrays
correlation clustering
treewidth computation
Bayesian network structure learning
causal discovery
visualization
model-based diagnosis
cutting planes for IPs
argumentation dynamics

[Park, 2002]
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
[Morgado and Marques-Silva, 2010]
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[Ignatiev, Janota, and Marques-Silva, 2014]

[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
[Lyne and Marques-Silva, 2011]
[Guerra and Lyne, 2012]
[Morgado, Liffiton, and Marques-Silva, 2012]
[Zhang and Bacchus, 2012]
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016a]
[Berg and Järvisalo, 2014]
[Berg, Järvisalo, and Malone, 2014]
[Hyttinen, Eberhardt, and Järvisalo, 2014]
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
[Saikko, Malone, and Järvisalo, 2015]
[Wallner, Niskanen, and Järvisalo, 2016]
Basic Concepts
**MaxSAT: Basic Definitions**

**MaxSAT**

**INPUT:** a set of clauses $F$. (a CNF formula)

**TASK:** find $\tau$ s.t. $\sum_{C \in F} \tau(C)$ is maximized.

Find a truth assignment that satisfies the maximum number of clauses

This is the standard definition:

- Much studied in theoretical computer science
- Often inconvenient for modeling practical problems.
Central Generalizations of MaxSAT

Weighted MaxSAT
- Each clause $C$ has an associated weight $w_C$
- Optimal solutions maximize the sum of weights of satisfied clauses

Partial MaxSAT
- Some clauses are deemed hard—infinite weights
  - Any solution has to satisfy the hard clauses
    - Existence of solutions not guaranteed
- Clauses with finite weight are soft

Weighted Partial MaxSAT
- Hard clauses (partial) + weights on soft clauses (weighted)
Terminology

- **Solution:**
  an assignment that satisfies all hard clauses

- **Cost of a solution:**
  the sum of weights of falsified soft clauses

- **Optimal solution:**
  minimizes cost over all solutions
Example: Encoding shortest paths

Shortest Path

Find shortest path in a grid with horizontal/vertical moves. Travel from S to G without entering blocked squares (black).

Note: best solved with state-space search

Here: to illustrate MaxSAT encodings
Example: Encoding shortest paths

Shortest Path

Find shortest path in a grid with horizontal/vertical moves. Travel from $S$ to $G$ without entering blocked squares (black).

Note: best solved with state-space search

Here: to illustrate $\text{MaxSAT}$ encodings
### MaxSAT: Example

<table>
<thead>
<tr>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>l</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>g</td>
<td>m</td>
</tr>
</tbody>
</table>

| S |

- **Boolean variables**: one for each unblocked grid square 
  \( \{S, G, a, b, \ldots, u\} \): true iff path visits this square.

- **Constraints**:
  - The S and G squares must be visited:
    - In CNF: unit hard clauses \((S)\) and \((G)\).
  - A soft clause of weight 1 for all other squares:
    - In CNF: \((\neg a), (\neg b), \ldots, (\neg u)\)
      - "would prefer not to visit"
MaxSAT: Example

- **Boolean variables**: one for each unblocked grid square 
  \{S, G, a, b, \ldots, u\}: true iff path visits this square.

- **Constraints**:
  - The S and G squares must be visited:
    In CNF: unit hard clauses (S) and (G).
  - A soft clause of weight 1 for all other squares:
    In CNF: (¬a), (¬b), \ldots, (¬u)
    “would prefer not to visit”
MaxSAT: Example

- The previous clauses minimize the number of visited squares.
- ...however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

A way to enforce a path between S and G:

- Both S and G must have exactly one visited neighbour
  - Any path starts from S
  - Any path ends at G
- Other visited squares must have exactly two visited neighbours
  - One predecessor, one successor on the path
**MaxSAT: Example**

- The previous clauses minimize the number of visited squares.
- . . . however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

### A way to enforce a path between S and G:

- Both S and G must have *exactly one* visited neighbour
  - Any path starts from S
  - Any path ends at G
- Other visited squares must have *exactly two* visited neighbours
  - One predecessor, one successor on the path

![Diagram of a grid with squares marked black and labels for S, G, etc.]
MaxSAT: Example

Constraint 1:

**S and G must have exactly one visited neighbour.**

- For **S**: \( a + b = 1 \)
  - In CNF:

- For **G**: \( k + q + r = 1 \)
  - “At least one” in CNF:
  - “At most one” in CNF:

\[
\begin{align*}
(a \lor b), (\neg a \lor \neg b) \\
(k \lor q \lor r), (\neg k \lor \neg q), (\neg k \lor \neg r), (\neg q \lor \neg r)
\end{align*}
\]

disallow pairwise
**MaxSAT: Example**

**Constraint 1:**

*\(S\) and \(G\) must have exactly one visited neighbour.*

- **For \(S\):** \(a + b = 1\)
  - In CNF:
  - \((a \lor b), (\neg a \lor \neg b)\)

- **For \(G\):** \(k + q + r = 1\)
  - “At least one” in CNF:
  - \((k \lor q \lor r)\)
  - “At most one” in CNF:
  - \((\neg k \lor \neg q), (\neg k \lor \neg r), (\neg q \lor \neg r)\)
  - *disallow pairwise*
MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

- **For S:** \( a + b = 1 \)
  - In CNF:
- **For G:** \( k + q + r = 1 \)
  - “At least one” in CNF:
  - “At most one” in CNF:

\[
(a \lor b), \neg a \lor \neg b
\]
\[
(k \lor q \lor r), \neg k \lor \neg q, \neg k \lor \neg r, \neg q \lor \neg r
\]

*disallow pairwise*
MaxSAT: Example

Constraint 2:  
*Other visited squares must have exactly two visited neighbours*

- For example, for square e:
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

Encoding Cardinality Constraints in CNF

- An important class of constraints, occur frequently in real-world problems
  - A lot of existing work on CNF encodings of cardinality constraints

---

<table>
<thead>
<tr>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>l</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>b</td>
<td>g</td>
<td>m</td>
</tr>
</tbody>
</table>
MaxSAT: Example

Constraint 2:
Other visited squares must have exactly two visited neighbours

- For example, for square e:
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

Encoding Cardinality Constraints in CNF

- An important class of constraints, occur frequently in real-world problems
  - A lot of existing work on CNF encodings of cardinality constraints
Properties of the encoding

- Every solution to the hard clauses is a path from $S$ to $G$ that does not pass a blocked square.
Properties of the encoding

- Every solution to the hard clauses is a path from $S$ to $G$ that does not pass a blocked square.
- Such a path will falsify one negative soft clause for every square it passes through.

- **orange path**: assign 14 variables in \{S, a, c, h, ... , t, r, G\} to true
MaxSAT: Example

Properties of the encoding

- Every solution to the hard clauses is a path from $S$ to $G$ that does not pass a blocked square.
- Such a path will falsify one negative soft clause for every square it passes through.
  - orange path: assign 14 variables in $\{S, a, c, h, \ldots, t, r, G\}$ to true
- MaxSAT solutions:
  - paths that pass through a minimum number of squares (i.e., is shortest).
  - green path: assign 8 variables in $\{S, b, g, f, \ldots, k, G\}$ to true
MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

Basic Idea

Finite-domain soft constraint \( C \) with associated weight \( W_C \).

Let \( \text{CNF}(C) = \bigwedge_{i=1}^{m} C_i \) be a CNF encoding of \( C \).

Softening \( \text{CNF}(C) \) as Weighted Partial MaxSAT:

- Hard clauses: \( \bigwedge_{i=1}^{m} (C_i \lor a) \), where \( a \) is a fresh Boolean variable
- Soft clause: \( (\neg a) \) with weight \( W_C \).
MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

**Basic Idea**

Finite-domain soft constraint $C$ with associated weight $W_C$.

Let $\text{CNF}(C) = \bigwedge_{i=1}^{m} C_i$ be a CNF encoding of $C$.

Softening $\text{CNF}(C)$ as Weighted Partial MaxSAT:

- **Hard clauses**: $\bigwedge_{i=1}^{m} (C_i \lor a)$,
  where $a$ is a fresh Boolean variable
- **Soft clause**: $(\neg a)$ with weight $W_C$. 
Representing High-Level Soft Constraints in MaxSAT

MaxSAT allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

Basic Idea

Finite-domain soft constraint $C$ with associated weight $W_C$.

Let $CNF(C) = \bigwedge_{i=1}^{m} C_i$ be a CNF encoding of $C$.

Softening $CNF(C)$ as Weighted Partial MaxSAT:

- Hard clauses: $\bigwedge_{i=1}^{m} (C_i \lor a)$,
  where $a$ is a fresh Boolean variable
- Soft clause: $(\neg a)$ with weight $W_C$.

Important for various applications of MaxSAT
MaxSAT: Complexity

Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.

**Question:** Is there an assignment that satisfies at least $k$ clauses in $F$?

MaxSAT is $\text{FP}^{\text{NP}}$–complete

- The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle
  - Polynomial number of oracle calls
  - Other $\text{FP}^{\text{NP}}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

MaxSAT is hard to approximate: APX–complete

APX: class of NP optimization problems that
- admit a constant-factor approximation algorithm, *but*
- have no poly-time approximation scheme (unless $\text{NP} = \text{P}$).
MaxSAT: Complexity

Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.

**Question:** Is there an assignment that satisfies at least $k$ clauses in $F$?

MaxSAT is FP$^\text{NP}$–complete

- The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle
  - Polynomial number of oracle calls
  - Other FP$^\text{NP}$–complete problems include TSP

- A SAT solver acts as the NP oracle most often in practice

MaxSAT is hard to approximate

APX: class of NP optimization problems that

- admit a constant-factor approximation algorithm, *but*
- have no poly-time approximation scheme (unless NP= P).
## MaxSAT: Complexity

<table>
<thead>
<tr>
<th>Deciding whether $k$ clauses can be satisfied: NP-complete</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A CNF formula $F$, a positive integer $k$.</td>
</tr>
<tr>
<td><strong>Question:</strong> Is there an assignment that satisfies at least $k$ clauses in $F$?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MaxSAT is $\text{FP}^{\text{NP}}$–complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle</td>
</tr>
<tr>
<td>- Polynomial number of oracle calls</td>
</tr>
<tr>
<td>- Other $\text{FP}^{\text{NP}}$–complete problems include TSP</td>
</tr>
<tr>
<td>- A SAT solver acts as the NP oracle most often in practice</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MaxSAT is hard to approximate</th>
</tr>
</thead>
<tbody>
<tr>
<td>APX–complete</td>
</tr>
<tr>
<td>APX: class of NP optimization problems that</td>
</tr>
<tr>
<td>- admit a constant-factor approximation algorithm, <em>but</em></td>
</tr>
<tr>
<td>- have no poly-time approximation scheme (unless NP=P).</td>
</tr>
</tbody>
</table>
Practical MaxSAT Solving
Standard Solver Input Format: DIMACS WCNF

- Variables indexed from 1 to $n$
- Negation: $\neg$
  - $-3$ stand for $\neg x_3$
- 0: special end-of-line character
- One special header “p”-line:
  
  ```
  p wcnf <#vars> <#clauses> <top>
  ```
  - #vars: number of variables $n$
  - #clauses: number of clauses
  - top: “weight” of hard clauses.
    - Any number larger than the sum of soft clause weights can be used.
- Clauses represented as lists of integers
  - Weight is the first number
  - $(\neg x_3 \lor x_1 \lor \neg x_{45})$, weight 2:
    
    ```
    2 -3 1 -45 0
    ```
- Clause is hard if weight $==$ top

---

Example:

```plaintext
mancoosi-test-i2000d0u98-26.wcnf
p wcnf 18169 112632 31540812410
31540812410 -1 2 3 0
31540812410 -4 2 3 0
31540812410 -5 6 0
... truncated 2.4 MB
```
MaxSAT Evaluations

Objectives

- Assessing the state of the art in the field of Max-SAT solvers
- Creating a collection of publicly available Max-SAT benchmark instances
- Tens of solvers from various research groups internationally participate each year
- Standard input format

11th MaxSAT Evaluation
http://maxsat.ia.udl.cat

Affiliated with SAT 2016: 19th Int’l Conference on Theory and Applications of Satisfiability Testing
Push-Button Solvers

- Black-box, no command line parameters necessary
- Input: CNF formula, in the standard DIMACS WCNF file format
- Output: provably optimal solution, or UNSATISFIABLE
  - Complete solvers

Internally rely especially on CDCL SAT solvers for proving unsatisfiability of subsets of clauses

---

mancoosi-test-i2000d0u98-26.wcnf

```
p wcnf 18169 112632 31540812410
31540812410 -1 2 3 0
31540812410 -4 2 3 0
31540812410 -5 6 0
...
18170 1133 0
18170 457 0
... truncated 2.4 MB
```
**Push-Button Solver Technology**

**Example:** $ openwbo mancoosi-test-i2000d0u98-26.wcnf

```
c Open-WBO: a Modular MaxSAT Solver
c Version: 1.3.1 – 18 February 2015

...c — Problem Type: Weighted
c — Number of variables: 18169
c — Number of hard clauses: 94365
c — Number of soft clauses: 18267
c — Parse time: 0.02 s
```

```
...c Relaxed soft clauses 2 / 18267
c LB : 15026590
```

```
...c Relaxed soft clauses 2 / 18267
c LB : 30053180
c Relaxed soft clauses 3 / 18267
```

```
c LB : 45079770
c Relaxed soft clauses 5 / 18267
```

```
c LB : 60106360
```

```
...c Relaxed soft clauses 726 / 18267
c LB : 287486453
```

```
c Relaxed soft clauses 728 / 18267
```

```
o 10548793370
```

```
c LB : 15026590
```

```
c LB : 30053180
```

```
c LB : 45079770
```

```
c LB : 60106360
```

```
... -18167 -18168 -18169 -18170
```

```
OPTIMUM FOUND
```

Relevant commands or outputs include:

- `$ openwbo mancoosi-test-i2000d0u98-26.wcnf`
- Parse time: 0.02 s
- Total time: 1.30 s
- OPTIMUM FOUND
- Variations and relaxation of soft clauses with LB values.
Progress in MaxSAT Solver Performance

Comparing some of the best solvers from 2010–2014:

In 2014: 50% more instances solved than in 2010!
Some Recent MaxSAT Solvers

Open-source:
- OpenWBO: http://sat.inesc-id.pt/open-wbo/
- MaxHS: http://maxhs.org
- LMHS: http://www.cs.helsinki.fi/group/coreo/lmhs/

Binaries available:
- Eva: http://www.maxsat.udl.cat/14/solvers/eva500a_
- MSCG: http://sat.inesc-id.pt/~aign/soft/
- WPM3: http://web.udl.es/usuarios/q4374304/#software
- QMaxSAT: https://sites.google.com/site/qmaxsat/
Algorithms for MaxSAT Solving
A Variety of Approaches

Branch and bound:
- ahmaxsat: http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html

Direct Integer Programming (IP) Encoding

Iterative, model-based:
- QMaxSAT: https://sites.google.com/site/qmaxsat/

Core-based:
- Eva: http://www.maxsat.udl.cat/14/solvers/eva500a__
- MSCG: http://sat.inesc-id.pt/~aign/soft/
- OpenWBO: http://sat.inesc-id.pt/open-wbo/
- WPM: http://web.udl.es/usuarios/q4374304/#software
- maxino: http://alviano.net/software/maxino/

IP-SAT Hybrids:
- MaxHS: http://maxhs.org
- LMHS: http://www.cs.helsinki.fi/group/coreo/lmhs/
Branch and Bound
Branch and Bound

- $UB =$ cost of the best solution so far.
- $mincost(n)$
  = minimum cost achievable under node $n$
- Backtrack when $mincost(n) \geq UB$
  - No solution under $n$ can improve $UB$.
- Goal:
  compute a lower bound $LB$ s.t.
  $mincost(n) \geq LB$.
- When $LB \geq UB$:
  $mincost(n) \geq LB \geq UB$
  $\sim \rightarrow$ backtrack.
Lower Bounds by Cores

**Common LB technique in MaxSAT solvers:**

Look for inconsistencies that force some soft clause to be falsified.
Lower Bounds by Cores

Common LB technique in $\text{MaxSAT}$ solvers:

Look for inconsistencies that force some soft clause to be falsified.

Example. $F = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots$

Ignoring clause costs, $\kappa = \{(x), (\neg x)\}$ is unsatisfiable.
Lower Bounds by Cores

Common LB technique in MaxSAT solvers:
Look for inconsistencies that force some soft clause to be falsified.

**Example.** $F = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots$
Ignoring clause costs, $\kappa = \{(x), (\neg x)\}$ is unsatisfiable.

Let $\kappa' = \{(*, 2), (\neg x, 1)\}$.

- Then $\kappa'$ is MaxSAT-equivalent to $\kappa$:
  the cost of each truth assignment is preserved.
Lower Bounds by Cores

Common LB technique in **MaxSAT** solvers:
Look for inconsistencies that force some soft clause to be falsified.

**Example.** $F = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots$

Ignoring clause costs, $\kappa = \{(x), (\neg x)\}$ is unsatisfiable.

Let $\kappa' = \{(\emptyset, 2), (\neg x, 1)\}$.

- Then $\kappa'$ is **MaxSAT-equivalent** to $\kappa$:
  
  the cost of each truth assignment is preserved.

Let $F' = (F \setminus \kappa) \cup \kappa'$. $F'$ is **MaxSAT-equivalent** to $F$.

The cost of $\emptyset$ has been incremented by 2

- Cost of $(\emptyset, 2)$ must be incurred: **2 is a LB**
Lower Bounds

1. Detect an unsatisfiable subset $\kappa$ of clauses (aka core) of the current formula
   
   - e.g. $\kappa = \{(x, 2) \land (\neg x, 3)\}$

2. Apply sound transformation to the clauses in $\kappa$ that result in an increment to the cost of the empty clause $\emptyset$
   
   - e.g. $\kappa$ replaced by $\kappa' = \{\emptyset, 2) \land (\neg x, 1)\}$
   
   - This replacement increases cost of $\emptyset$ by 2.

3. Repeat 1 and 2 until no $LB$ cannot be incremented (or $LB \geq UB$)
Fast Detection of Cores by UP

Treat the soft clauses as if they were hard and then:

- Run **Unit Propagation** (UP).
  If UP falsifies a clause we can find a core.
  
  **Example.** On \{(x, 2), (\neg x, 3)\} UP falsified a clause.

- The falsified clause and the clauses that generated it form a core.
- This can find inconsistent sub-formulas quickly.
  But only *some* inconsistent sub-formulas.
Transforming the Formula

Various sound transformations of cores into increments of the empty clause have been identified.

- **MaxRes** generalizes this to provide a sound and complete inference rule for **MaxSAT**

  [Larrosa and Heras, 2005]
  [Bonet, Levy, and Manyà, 2007]

- **Other Lower Bounding Techniques**
  - Falsified soft learnt clauses and hitting sets over their proofs
    [Davies, Cho, and Bacchus, 2010]
  - Minibuckets, width-restricted BDDs
    [Dechter and Rish, 2003]
    [Bergman, Ciré, van Hoeve, and Yunes, 2014]
Branch-and-Bound: Summary

- **Strengths:**
  Can be effective on small combinatorially hard problems, e.g., maxclique in a graph.

- **Weaknesses:**
  Once the number of variables gets to 1,000 or more it is less effective: LB techniques become weak or too expensive.
MaxSAT by Integer Programming (IP)
Solving MaxSAT with an IP Solver

Optimization problems studied for decades in Operations Research

IP solvers the most common optimization tool in OR.
- IBM CPLEX, Gurobi, SCIP, ...

- IP solvers solve problems with linear constraints and objective function where some variables are integers.
- Branch-and-cut solver algorithms, essentially:
  - Compute a series of linear relaxations and cuts
    (new linear constraints that cut off non-integral solutions).
  - Sometimes branch on a bound for an integer variable.

- State-of-the-art IP solvers very powerful and effective:
  at times also for solving MaxSAT instances!
Solving MaxSAT with an IP Solver

Optimization problems studied for decades in Operations Research
IP solvers the most common optimization tool in OR.
- IBM CPLEX, Gurobi, SCIP, ...

- IP solvers solve problems with linear constraints and objective function where some variables are integers.
- Branch-and-cut solver algorithms, essentially:
  - Compute a series of linear relaxations and cuts (new linear constraints that cut off non-integral solutions).
  - Sometimes branch on a bound for an integer variable.

- State-of-the-art IP solvers very powerful and effective: at times also for solving MaxSAT instances!
Relaxing Clauses

MaxSAT algorithms frequently use relaxation (selector, blocking, . . .) variables to relax soft clauses.

- Given a soft clause \((x_1 \lor x_2 \lor \cdots \lor x_k)\):
  - add a **new** variable \(r\) to obtain

\[
(r \lor x_1 \lor x_2 \lor \cdots \lor x_k)
\]

*note: \(r\) does not appear anywhere else in the formula*

- If \(r = 1\): the soft clause is automatically satisfied
  (relaxed, switched off).

- If \(r = 0\): the clause becomes hard and must be satisfied
  (switched on).
MaxSAT encoding into IP

1. For each soft clause $C_i$, relax $C_i$ by augmenting it with a new relaxation variable $r_i$.

\[(x \lor \neg y \lor z \lor \neg w) \sim (r_i \lor x \lor \neg y \lor z \lor \neg w)\]

2. Convert every augmented clause into a linear constraint:

\[r_i + x + (1 - y) + z + (1 - w) \geq 1\]

3. Boolean variables: bound integer domains to \{0, 1\}

4. Objective function:

\[
\text{minimize } \sum_{C_i \in F_s} r_i \cdot w_i,
\]

where $w_i$ is the weight of the soft clause $C_i \in F_s$
Integer Programming Summary

- IP solvers use Branch and Cut
  - Compute a series of linear relaxations and cuts: new linear constraints that cut off non-integral solutions.
  - Sometimes branch on a bound for an integer variable.
  - (And several other techniques)

- Effective on many standard optimization problems.
- Do not (always) dominate “native” MaxSAT solvers on “very Boolean” problem classes
SAT-Based $\text{MAXSAT}$ Solving
SAT-Based MaxSAT Solving

- Solve a sequence or SAT instances where each instance encodes a decision problem of the form

  “Is there a truth assignment of falsifying at most weight \( k \) soft clauses?”

for different values of \( k \).

- SAT-based MaxSAT algorithms mainly do two things:
  1. Develop better ways to encode this decision problem.
  2. Find ways to exploit information obtained from the SAT solver at each stage in the next stage.

Assume unit weight soft clauses for now
SAT-Based **MaxSAT** Solving

- Iterative search methods
- Improving by using cores
- Recent advances
Iterative Search

Basic approach:

- To check whether $F$ has a solution of cost $\leq k$:
  - SAT solve $(C_1 \lor r_1) \land (C_2 \lor r_2) \land \cdots \land (C_n \lor r_n) \land (\sum_{i=1}^{n} r_i \leq k)$

- Iterate over $k \in \{1, \ldots, n\}$ to find the optimal $k$
  - ...and an optimal solution.
  - ...proving that no solutions of cost $< k$ exist.
Iterative Search

Basic approach:

- To check whether $F$ has a solution of cost $\leq k$:
  - SAT solve $(C_1 \lor r_1) \land (C_2 \lor r_2) \land \cdots \land (C_n \lor r_n) \land (\sum_{i=1}^{n} r_i \leq k)$

- Iterate over $k \in \{1, \ldots, n\}$ to find the optimal $k$
  - ...and an optimal solution.
  - ...proving that no solutions of cost $< k$ exist.
Iterating over $k$

- Different ways of iterating over values of $k$.
- Three “standard” approaches:

1. Linear search UNSAT to SAT (not effective)
   - Start from $k = 1$.
   - Increment $k$ by 1 until a solution is found.

2. Binary search (effective with core-based reasoning)
   - $UB = \# \text{ of soft clauses}; \ LB = 0$.
   - Solve with $k = (UB + LB)/2$.
   - If SAT: $UB = k$; if UNSAT: $LB = k + 1$
   - When $UB = LB + 1$, $UB$ is solution.
Iterating over $k$

3. Linear search SAT to UNSAT
   1. Find a satisfying assignment $\pi$ of the hard clauses.
   2. Solve with $k = (\# \text{ of clauses falsified by } \pi) - 1$
   3. If SAT: found better assignment. Reset $k$ and repeat 2.
   4. If UNSAT: last assignment $\pi$ found is optimal.

- Finds a sequence of improved solutions
- Used in e.g. QMaxSAT, can be effective on certain problems
SAT-based MaxSAT Solving using Cores
Core-Based MaxSAT Solving

Motivation

- In the linear approach:
  - add $CNF(\sum r_i \leq k)$ to the SAT solver.
  - One $r_i$ per each soft clause.
  - The cardinality constraint could be over 100,000s of variables and is very loose:
    - No information about which relaxation variables to assign to 1

- This makes SAT solving inefficient:
  - could have to explore many choices of subsets of $k$ soft clauses to remove.

Obtaining an UNSAT core gives a more powerful constraints over which particular soft clauses to relax.
Core-Based **MaxSAT** Solving

**Motivation**

- In the linear approach:
  - add $CNF(\sum r_i \leq k)$ to the SAT solver.
    - One $r_i$ per each soft clause.
    - The cardinality constraint could be over 100,000s of variables
      ... and is very loose:
      No information about which relaxation variables to assign to 1
  - This makes SAT solving inefficient:
    - could have to explore many choices of subsets of $k$ soft clauses to remove.

*Obtaining an *UNSAT* core gives a more powerful constraints over which particular soft clauses to relax.*
Unsatisfiable Cores in MaxSAT

**UNSAT core in MaxSAT**

A subset $F_s' \subseteq F_s$ such that $F_h \land F_s'$ is unsatisfiable.

- The hard clauses act as background theory
- ...but are *not* part of an UNSAT core

**Fact**

For each UNSAT core $F_s'$:

*some clause* $C \in F_s'$ need to be removed to make $F_h \land F_s'$ satisfiable.

- That is: at least one clause from every core must be left unsatisfied.

**Core-based constraints**

- Instead of iteratively ruling out non-optimal solutions: *iteratively find and rule out UNSAT cores.*
- Core-based vs cardinality constraints over *all* soft clauses:
  - Typically cores are *much* smaller than the set of all soft clauses.
Unsatisfiable Cores in MaxSAT

**UNSAT core in MaxSAT**

A subset $F'_s \subseteq F_s$ such that $F_h \land F'_s$ is unsatisfiable.

- The hard clauses act as background theory
- ...but are *not* part of an UNSAT core

**Fact**

For each UNSAT core $F'_s$:

*some clause $C \in F'_s$ need to be removed to make $F_h \land F'_s$ satisfiable.*

- That is: at least one clause from every core must be left unsatisfied.

**Core-based constraints**

- Instead of iteratively ruling out non-optimal solutions: *iteratively find and rule out UNSAT cores.*
- Core-based vs cardinality constraints over *all* soft clauses:
  - Typically cores are *much* smaller than the set of all soft clauses.
Unsatisfiable Cores in MaxSAT

**UNSAT core in MaxSAT**

A subset $F'_s \subseteq F_s$ such that $F_h \land F'_s$ is unsatisfiable.

- The hard clauses act as background theory
- ...but are *not* part of an UNSAT core

**Fact**

For each UNSAT core $F'_s$:

*some clause* $C \in F'_s$ need to be removed to make $F_h \land F'_s$ satisfiable.

- That is: at least one clause from every core must be left unsatisfied.

**Core-based constraints**

- Instead of iteratively ruling out non-optimal solutions: *iteratively find and rule out UNSAT cores.*
- Core-based vs cardinality constraints over *all* soft clauses:
  - Typically cores are *much* smaller than the set of all soft clauses.
Core-Guided MaxSAT Algorithms: Fu-Malik

The first core-guided MaxSAT algorithm

[Fu and Malik, 2006]

Fu-Malik Algorithm

Iteratively:

- Find an UNSAT core using a SAT solver
- Add relaxation variables to clauses in the core
- Add an AtMost-1 constraint over the new relaxation variables
  ▶ Soft clauses remain soft after relaxing them

...until the SAT solver reports *satisfiable*.

Key observation

Each iteration lowers the cost of solutions by 1 (on an unweighted formula)
The first core-guided MaxSAT algorithm

Fu-Malik Algorithm

Iteratively:
- Find an UNSAT core using a SAT solver
- Add relaxation variables to clauses in the core
- Add an AtMost-1 constraint over the new relaxation variables
  - Soft clauses remain soft after relaxing them

... until the SAT solver reports *satisfiable*.

Key observation

Each iteration *lowers the cost of solutions by 1* (on an unweighted formula)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \quad \quad C_2 = \neg x_6 \lor x_2 \quad \quad C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_4 = \neg x_1 \lor r_2 \quad \quad C_5 = \neg x_6 \lor x_8 \quad \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \quad \quad C_8 = \neg x_4 \lor x_5 \lor r_4 \quad \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad \quad C_{11} = \neg x_5 \lor x_3 \lor r_5 \quad \quad C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.
Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \lor r_2 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.
Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \lor r_2 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Fu-Malik: Example
(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \lor r_7 \]
\[ C_4 = \neg x_1 \lor r_2 \lor r_{10} \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \lor r_{12} \]
\[ C_2 = \neg x_6 \lor x_2 \lor r_8 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \lor r_{13} \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \lor r_9 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_{12} = \neg x_3 \lor r_6 \lor r_{14} \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \lor r_7 \]
\[ C_4 = \neg x_1 \lor r_2 \lor r_{10} \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \lor r_{12} \]
\[ C_2 = \neg x_6 \lor x_2 \lor r_8 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \lor r_{13} \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \lor r_9 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_9 = x_7 \lor x_5 \lor r_{11} \]
\[ C_{12} = \neg x_3 \lor r_6 \lor r_{14} \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
4. UNSAT core: \{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{14}\)
6. Add \(\sum_{i=7}^{14} r_i \leq 1\)
7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Fu-Malik: Example

(On an unweighted formula)

\begin{align*}
C_1 &= x_6 \lor x_2 \lor r_7 \\
C_4 &= \neg x_1 \lor r_2 \lor r_{10} \\
C_7 &= x_2 \lor x_4 \lor r_3 \\
C_{10} &= \neg x_7 \lor x_5 \lor r_{12} \\
C_2 &= \neg x_6 \lor x_2 \lor r_8 \\
C_5 &= \neg x_6 \lor x_8 \\
C_8 &= \neg x_4 \lor x_5 \lor r_4 \\
C_{11} &= \neg x_5 \lor x_3 \lor r_5 \lor r_{13} \\
C_3 &= \neg x_2 \lor x_1 \lor r_1 \lor r_9 \\
C_6 &= x_6 \lor \neg x_8 \\
C_9 &= x_7 \lor x_5 \lor r_{11} \\
C_{12} &= \neg x_3 \lor r_6 \lor r_{14}
\end{align*}

1. UNSAT core: \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \}
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \)
4. UNSAT core: \{ C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12} \}
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{14} \)
6. Add \( \sum_{i=7}^{14} r_i \leq 1 \)
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

MSU3 is a another MaxSAT algorithm for exploiting cores

[Marques-Silva and Planes, 2007].

Differences to Fu-Malik:

- Introduce only at most one relaxation variable to each soft clause
  - Re-use already introduced relaxation variables

- Instead of adding one AtMost-1/Exactly-1 constraint per iteration: *Update* the AtMost-\(k\), \(k\) noting the \(k\)th iteration

- Relaxed soft clauses *become hard*
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \) \hspace{1cm} \text{AtMost-}k \text{ where } k = 1
4. UNSAT core: \{C_1, C_2, C_9, C_{10}\}
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \) \hspace{1cm} \text{AtMost-}k \text{ where } k = 2
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \) \quad AtMost-\( k \) where \( k = 1 \)
4. UNSAT core: \{C_1, C_2, , C_9, C_{10}\}
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \) \quad AtMost-\( k \) where \( k = 2 \)
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \lor r_2 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \)
4. UNSAT core: \{C_1, C_2, , C_9, C_{10}\}
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \)
7. Satisfiable, terminate.
Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[

c_1 = x_6 \lor x_2 \\
c_4 = \neg x_1 \lor r_2 \\
c_7 = x_2 \lor x_4 \lor r_3 \\
c_{10} = \neg x_7 \lor x_5
\]

\[

c_2 = \neg x_6 \lor x_2 \\
c_5 = \neg x_6 \lor x_8 \\
c_8 = \neg x_4 \lor x_5 \lor r_4 \\
c_{11} = \neg x_5 \lor x_3 \lor r_5
\]

\[

c_3 = \neg x_2 \lor x_1 \lor r_1 \\
c_6 = x_6 \lor \neg x_8 \\
c_9 = x_7 \lor x_5 \\
c_{12} = \neg x_3 \lor r_6
\]

1. UNSAT core: \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}

2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)

3. Add \(\sum_{i=1}^{6} r_i \leq 1\) \quad AtMost-k \quad \text{where} \quad k = 1

4. UNSAT core: \{c_1, c_2, , c_9, c_{10}\}

5. Relax the clauses in the core with variables \(r_7, \ldots, r_{10}\)

6. Update the AtMost-1 to: \(\sum_{i=1}^{10} r_i \leq 2\) \quad AtMost-k \quad \text{where} \quad k = 2

7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \lor r_2 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \( \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \)
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \)
   \[ \text{AtMost-}k \text{ where } k = 1 \]
4. UNSAT core: \( \{ C_1, C_2, C_9, C_{10} \} \)
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \)
   \[ \text{AtMost-}k \text{ where } k = 2 \]
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[
C_1 = x_6 \lor x_2 \lor r_7 \\
C_4 = \neg x_1 \lor r_2 \\
C_7 = x_2 \lor x_4 \lor r_3 \\
C_{10} = \neg x_7 \lor x_5 \lor r_{12}
\]

\[
C_2 = \neg x_6 \lor x_2 \lor r_8 \\
C_5 = \neg x_6 \lor x_8 \\
C_8 = \neg x_4 \lor x_5 \lor r_4 \\
C_{11} = \neg x_5 \lor x_3 \lor r_5
\]

\[
C_3 = \neg x_2 \lor x_1 \lor r_1 \\
C_6 = x_6 \lor \neg x_8 \\
C_9 = x_7 \lor x_5 \lor r_{11} \\
C_{12} = \neg x_3 \lor r_6
\]

1. UNSAT core: \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}
2. Relax the clauses in the core with variables \(r_1, \ldots, r_6\)
3. Add \(\sum_{i=1}^{6} r_i \leq 1\)
   \[\text{AtMost-}k \text{ where } k = 1\]
4. UNSAT core: \{C_1, C_2, C_9, C_{10}\}
5. Relax the clauses in the core with variables \(r_7, \ldots, r_{10}\)
6. Update the AtMost-1 to: \(\sum_{i=1}^{10} r_i \leq 2\)
   \[\text{AtMost-}k \text{ where } k = 2\]
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[
\begin{align*}
C_1 &= x_6 \lor x_2 \lor r_7 \\
C_4 &= \neg x_1 \lor r_2 \\
C_7 &= x_2 \lor x_4 \lor r_3 \\
C_{10} &= \neg x_7 \lor x_5 \lor r_{12}
\end{align*}
\]

\[
\begin{align*}
C_2 &= \neg x_6 \lor x_2 \lor r_8 \\
C_5 &= \neg x_6 \lor x_8 \\
C_8 &= \neg x_4 \lor x_5 \lor r_4 \\
C_{11} &= \neg x_5 \lor x_3 \lor r_5
\end{align*}
\]

\[
\begin{align*}
C_3 &= \neg x_2 \lor x_1 \lor r_1 \\
C_6 &= x_6 \lor \neg x_8 \\
C_9 &= x_7 \lor x_5 \lor r_{11} \\
C_{12} &= \neg x_3 \lor r_6
\end{align*}
\]

1. UNSAT core: \( \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \)
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \) AtMost-\( k \) where \( k = 1 \)
4. UNSAT core: \( \{ C_1, C_2, C_9, C_{10} \} \)
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \) AtMost-\( k \) where \( k = 2 \)
7. Satisfiable, terminate.

Optimal cost: 2 (the number of iterations)
Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

\[ C_1 = x_6 \lor x_2 \lor r_7 \]
\[ C_4 = \neg x_1 \lor r_2 \]
\[ C_7 = x_2 \lor x_4 \lor r_3 \]
\[ C_{10} = \neg x_7 \lor x_5 \lor r_{12} \]
\[ C_2 = \neg x_6 \lor x_2 \lor r_8 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \lor r_4 \]
\[ C_{11} = \neg x_5 \lor x_3 \lor r_5 \]
\[ C_3 = \neg x_2 \lor x_1 \lor r_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \lor r_{11} \]
\[ C_{12} = \neg x_3 \lor r_6 \]

1. UNSAT core: \( \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \)
2. Relax the clauses in the core with variables \( r_1, \ldots, r_6 \)
3. Add \( \sum_{i=1}^{6} r_i \leq 1 \) \hspace{1cm} AtMost-\( k \) where \( k = 1 \)
4. UNSAT core: \( \{ C_1, C_2, , C_9, C_{10} \} \)
5. Relax the clauses in the core with variables \( r_7, \ldots, r_{10} \)
6. Update the AtMost-1 to: \( \sum_{i=1}^{10} r_i \leq 2 \) \hspace{1cm} AtMost-\( k \) where \( k = 2 \)
7. Satisfiable, terminate.
   Optimal cost: 2 (the number of iterations)
Some Further Core-based Ideas

- OpenWBO uses MSU3 with incremental cardinality constraints to achieve state-of-the-art performance on many problems.
  
  [Martins, Joshi, Manquinho, and Lynce, 2014]

  ▶ Combine with an incremental construction of the cardinality constraint: each new constraint builds on the encoding of the previous constraint

- WPM2 proposes a method for dealing with overlapping cores
  
  [Ansótegui, Bonet, and Levy, 2013a]

  ▶ Group intersecting cores into disjoint covers.

  The cores might not be disjoint but the covers will be

  ▶ at-most ≤ cardinality constraints over the soft clauses in a cover

  ▶ An at-least ≥ constraint over the clauses in a core

...
Recent Advances in Core-Based Algorithms  
(in short)
Recent Advances in Core-Based **MaxSAT** Solving

**Key Ideas**

- *Transform* the logical structure of the current formula
  - not only encode new cardinality constraints over relaxed clauses
- Use *soft* cardinality constraints
  - New logical encodings

- Currently some of the best SAT-based approaches EVA, MSCG-OLL, OpenWBO, WPM3, MAXINO

  [Narodytska and Bacchus, 2014]
  [Morgado, Dodaro, and Marques-Silva, 2014]
  [Martins, Joshi, Manquinho, and Lynce, 2014]
  [Ansótegui, Didier, and Gabàs, 2015]
  [Alviano, Dodaro, and Ricca, 2015]

**Central Research Question**

Achieve a better understand of the impact of these transformations on the SAT solving process
Dealing with Weighted Soft Clauses

How to deal with soft clauses with different weights?
Clause Cloning

Methor used to deal with varying weights

[Ansótegui, Bonet, and Levy, 2009; Manquinho, Silva, and Planes, 2009]

$K$ is new core.

$w_{\text{min}}$ is minimum weight in $K$.

1. Split each clause $(c, w) \in K$ into two clauses:
   (1) $(c, w_{\text{min}})$ and (2) $(c, w - w_{\text{min}})$.

2. Keep all clauses (2) $(c, w - w_{\text{min}})$ as soft clauses (discard zero weight clauses)

3. Let $K$ be all clauses (1) $(c, w_{\text{min}})$

4. Process $K$ as a new core
   (all clauses in $K$ have the same weight)
SAT-Based **MaxSAT**: Summary

- Effective on large MaxSAT instance
  - Especially when there are many hard clauses

- Central innovations:
  - Efficient ways to encode and solve the individual SAT decision problems that have to be solved.
  - Some work done on understanding the core structure and its impact on SAT solving efficiency but more needed.

[Bacchus and Narodytska, 2014]
[Berg and Järvisalo, 2016b]
Implicit Hitting Set Algorithms for MaxSAT

[Davies and Bacchus, 2011, 2013b,a]
Hitting Sets and UNSAT Cores

Hitting Sets

Given a collection $S$ of sets of elements, a set $H$ is a hitting set of $S$ if $H \cap S \neq \emptyset$ for all $S \in S$.

A hitting set $H$ is optimal if no $H' \subseteq \bigcup S$ with $|H'| < |H|$ is a hitting set of $S$.

Note: Under weight function $c : S \rightarrow \mathbb{R}^+$, $c(H') < c(H)$ where $c(H) = \sum_{h \in H} c(h)$.

What does this have to do with MaxSAT?

For any MaxSAT instance $F$:
for any optimal hitting set $H$ of the set of UNSAT cores of $F$, there is an optimal solutions $\tau$ to $F$ such that $\tau$ satisfies exactly the clauses $F \setminus H$. 

Järvisalo (U Helsinki)
Hitting Sets and UNSAT Cores

Hitting Sets
Given a collection $S$ of sets of elements,
A set $H$ is a hitting set of $S$ if $H \cap S \neq \emptyset$ for all $S \in S$.
A hitting set $H$ is optimal if no $H' \subset \bigcup S$ with $|H'| < |H|$ is a hitting set of $S$.

- Note: Under weight function $c : S \rightarrow \mathbb{R}^+$,
  $c(H') < c(H)$ where $c(H) = \sum_{h \in H} c(h)$.

What does this have to do with MaxSAT?
For any MaxSAT instance $F$:
for any optimal hitting set $H$ of the set of UNSAT cores of $F$,
there is an optimal solutions $\tau$ to $F$ such that $\tau$ satisfies exactly the clauses $F \setminus H$.
Hitting Sets and UNSAT Cores

Hitting Sets

Given a collection \( S \) of sets of elements, a set \( H \) is a hitting set of \( S \) if \( H \cap S \neq \emptyset \) for all \( S \in S \).

A hitting set \( H \) is optimal if no \( H' \subset \bigcup S \) with \( |H'| < |H| \) is a hitting set of \( S \).

Note: Under weight function \( c : S \to \mathbb{R}^+ \),
\[
c(H') < c(H) \quad \text{where} \quad c(H) = \sum_{h \in H} c(h).
\]

What does this have to do with MaxSAT?

For any MaxSAT instance \( F \):
for any optimal hitting set \( H \) of the set of UNSAT cores of \( F \),
there is an optimal solutions \( \tau \) to \( F \) such that \( \tau \) satisfies exactly the clauses \( F \setminus H \).
Hitting Sets and UNSAT Cores

Key insight

To find an optimal solution to a MaxSAT instance $F$, it suffices to:

- Find an (implicit) hitting set $F$ of the UNSAT cores of $F$.  
  - Implicit refers to not necessarily having all MUSes of $F$.
- Find a solution to $F \setminus H$.  

Implicit Hitting Set Approach to MaxSAT

Iterate over the following steps:

- Accumulate a collection $\mathcal{K}$ of UNSAT cores using a SAT solver
- Find an optimal hitting set $H$ over $\mathcal{K}$, and rule out the clauses in $H$ for the next SAT solver call using an IP solver

... until the SAT solver returns satisfying assignment.

Hitting Set Problem as Integer Programming

$$\min \sum_{C \in \mathcal{K}} c(C) \cdot r_C$$
$$\text{subject to } \sum_{C \in \mathcal{K}} r_C \geq 1 \quad \forall \mathcal{K} \in \mathcal{K}$$

- $r_C = 1$ iff clause $C$ in the hitting set
- Weight function $c$: works also for weighted MaxSAT
Implicit Hitting Set Approach to \textit{MaxSAT}

Iterate over the following steps:

- Accumulate a collection $\mathcal{K}$ of UNSAT cores using a SAT solver.
- Find an optimal hitting set $H$ over $\mathcal{K}$, and \textit{rule out the clauses in $H$ for the next SAT solver call} using an IP solver.

\ldots until the SAT solver returns satisfying assignment.

### Hitting Set Problem as Integer Programming

\[
\begin{align*}
\min \sum_{C \in \cup \mathcal{K}} c(C) \cdot r_C \\
\text{subject to } \sum_{C \in K} r_C & \geq 1 \quad \forall K \in \mathcal{K}
\end{align*}
\]

- $r_C = 1$ iff clause $C$ in the hitting set.
- Weight function $c$: works also for weighted MaxSAT.
Implicit Hitting Set Approach to MaxSAT

Intuition: combine the main strengths of SAT and IP solvers

- **SAT solvers** are very good at **proving unsatisfiability**
  - Provide explanations for unsatisfiability in terms of cores
  - Instead of adding clauses to / modifying the input MaxSAT instance: each SAT solver call made on a *subset* of the clauses in the instance

- **IP solvers** at **optimization**
  - Instead of directly solving the input MaxSAT instance: solve a sequence of *simpler* hitting set problems over the cores

Instantiation of the implicit hitting set approach

[Moreno-Centeno and Karp, 2013]
Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \rightarrow \mathbb{R}^+$
Solving MaxSAT by SAT and Hitting Set Computations

Input:
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \mapsto \mathbb{R}^+$

1. Initialize

   $F_h, F_s$
   $hs := \emptyset$
   $\mathcal{K} := \emptyset$

   $\mathcal{K} := \mathcal{K} \cup \{K\}$

SAT solver

$F_h \land (F_s \setminus hs)$

IP solver

$$\min \sum_{C \in \mathcal{K}} c(C) \cdot r_C$$
$$\sum_{C \in \mathcal{K}} r_C \geq 1 \ \forall K \in \mathcal{K}$$

Optimal solution found

UNSAT core extraction

Min-cost Hitting Set
Solving \texttt{MaxSAT} by SAT and Hitting Set Computations

**Input:**
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \mapsto \mathbb{R}^+$

2. UNSAT core

\[ F_h, F_s \\
hs := \emptyset \\
\mathcal{K} := \emptyset \]

\[ \mathcal{K} := \mathcal{K} \cup \{K\} \]

\[ \text{IP solver} \]

\[ \min \sum_{C \in \mathcal{K}} c(C) \cdot r_C \]

\[ \sum_{C \in \mathcal{K}} r_C \geq 1 \ \forall K \in \mathcal{K} \]

Optimal solution found
Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \mapsto \mathbb{R}^+$

3. Update core set

$F_h, F_s$
$hs := \emptyset$
$\mathcal{K} := \emptyset$

$\mathcal{K} := \mathcal{K} \cup \{K\}$

**SAT solver**

$F_h \land (F_s \setminus hs)$

**IP solver**

$\min \sum_{C \in \mathcal{K}} c(C) \cdot r_C$
$\sum_{C \in \mathcal{K}} r_C \geq 1 \ \forall K \in \mathcal{K}$

**UNSAT core extraction**

**Min-cost Hitting Set**

Optimal solution found
Solving Maximum Satisfiability (MaxSAT) by SAT and Hitting Set Computations

**Input:**
hard clauses $F_h$, soft clauses $F_s$, weight function $c: F_s \rightarrow \mathbb{R}^+$

1. **SAT solver**
   - Input: $F_h \wedge (F_s \setminus hs)$
   - Output: $sat$ or $unsat$

2. **UNSAT core extraction**
   - If $sat$, extract unsatisfiable core $hs$.
   - Else, continue.

3. **Min-cost HS**
   - $F_h, F_s$
   - $hs := \emptyset$
   - $K := \emptyset$
   - $K := K \cup \{K\}$

4. **IP solver**
   - $c$
   - $\min \sum_{C \in K} c(C) \cdot r_C$
   - $\sum_{C \in K} r_C \geq 1 \ \forall K \in K$

5. **Optimal solution found**
Solving \textsc{MaxSAT} by SAT and Hitting Set Computations

\textbf{Input:}
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \rightarrow \mathbb{R}^+$

5. UNSAT core

\[
\begin{align*}
F_h, F_s \\
hs := \emptyset \\
\mathcal{K} := \emptyset \\
\mathcal{K} := \mathcal{K} \cup \{K\} \\
\text{IP solver} \\
\min \sum_{C \in \mathcal{K}} c(C) \cdot r_C \\
\sum_{C \in K} r_C \geq 1 \ \forall K \in \mathcal{K}
\end{align*}
\]

\textbf{UNSAT core extraction} \quad \textbf{Min-cost Hitting Set}

Optimal solution found
Input:
hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \mapsto \mathbb{R}^+$

iterate until "sat"

$F_h, F_s$

$hs := \emptyset$

$K := \emptyset$

$K := K \cup \{K\}$

unsat

IP solver

$\min \sum_{C \in K} c(C) \cdot r_C$

$\sum_{C \in K} r_C \geq 1 \ \forall K \in K$

unsat

$F_h \wedge (F_s \setminus hs)$

Optimal solution found

Järvisalo (U Helsinki) | MaxSAT | June 22
Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses \( F_h \), soft clauses \( F_s \), weight function \( c : F_s \mapsto \mathbb{R}^+ \)

iterate until “sat”

\[
\begin{align*}
F_h, F_s \\
hs := \emptyset \\
\mathcal{K} := \emptyset \\
\mathcal{K} := \mathcal{K} \cup \{K\} \\
\text{IP solver} \\
\min \sum_{C \in \mathcal{K}} c(C) \cdot r_C \\
\sum_{C \in \mathcal{K}} r_C \geq 1 \quad \forall K \in \mathcal{K}
\end{align*}
\]

Optimal solution found
Solving \textsc{MaxSAT} by SAT and Hitting Set Computations

\textbf{Intuition:} After \textit{optimally} hitting all cores of \( F_h \land F_s \) by \( hs \): any solution to \( F_h \land (F_s \setminus hs) \) is \textit{guaranteed to be optimal}.

iterate until “sat”

\[
\begin{align*}
F_h, F_s \\
hs &:= \emptyset \\
\mathcal{K} &:= \emptyset \\
\mathcal{K} &:= \mathcal{K} \cup \{K\} \\
\text{IP solver} & \quad \min \sum_{C \in \mathcal{K}} c(C) \cdot r_C \\
& \quad \sum_{C \in \mathcal{K}} r_C \geq 1 \forall K \in \mathcal{K}
\end{align*}
\]

\text{Optimal solution found}

\text{Sat solver}

\( F_h \land (F_s \setminus hs) \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]
\[ \mathcal{K} := \emptyset \]
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]

\[ \mathcal{K} := \emptyset \]

- SAT solve \( F_h \land (F_s \setminus \emptyset) \)
**MaxSAT by SAT and Hitting Set Computation: Example**

\[
\begin{align*}
    C_1 &= x_6 \lor x_2 & C_2 &= \neg x_6 \lor x_2 & C_3 &= \neg x_2 \lor x_1 \\
    C_4 &= \neg x_1 & C_5 &= \neg x_6 \lor x_8 & C_6 &= x_6 \lor \neg x_8 \\
    C_7 &= x_2 \lor x_4 & C_8 &= \neg x_4 \lor x_5 & C_9 &= x_7 \lor x_5 \\
    C_{10} &= \neg x_7 \lor x_5 & C_{11} &= \neg x_5 \lor x_3 & C_{12} &= \neg x_3
\end{align*}
\]

\[
\mathcal{K} := \emptyset
\]

- SAT solve \( F_h \land (F_s \setminus \emptyset) \sim\) UNSAT core \( K = \{C_1, C_2, C_3, C_4\}\)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \} \} \]

- Update \( \mathcal{K} := \mathcal{K} \cup \{ K \} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \} \} \]

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{\{C_1, C_2, C_3, C_4\}\} \]

- Solve minimum-cost hitting set problem over \( \mathcal{K} \sim hs = \{C_1\} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \} \} \]

- SAT solve \( F_h \land (F_s \setminus \{ C_1 \}) \)
MaxSAT by SAT and Hitting Set Computation: Example

\[
C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \\
C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \\
C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \\
C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3
\]

\[
\mathcal{K} := \{ C_1, C_2, C_3, C_4 \}
\]

- SAT solve $F_h \land (F_s \setminus \{ C_1 \}) \leadsto$ UNSAT core $K = \{ C_9, C_{10}, C_{11}, C_{12} \}$
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{\{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \}\} \]

- Update \( \mathcal{K} := \mathcal{K} \cup \{ K \} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \} \} \]

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
**MaxSAT by SAT and Hitting Set Computation: Example**

\[C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1\]
\[C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8\]
\[C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5\]
\[C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3\]

\[\mathcal{K} := \{\{C_1, C_2, C_3, C_4\}, \{C_9, C_{10}, C_{11}, C_{12}\}\}\]

- Solve minimum-cost hitting set problem over \(\mathcal{K} \leadsto hs = \{C_1, C_9\}\)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{\{C_1, C_2, C_3, C_4\}, \{C_9, C_{10}, C_{11}, C_{12}\}\} \]

- SAT solve \( F_h \land (F_s \setminus \{C_1, C_9\}) \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad \quad C_2 = \neg x_6 \lor x_2 \quad \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad \quad C_5 = \neg x_6 \lor x_8 \quad \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad \quad C_8 = \neg x_4 \lor x_5 \quad \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad \quad C_{11} = \neg x_5 \lor x_3 \quad \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \} \} \]

- SAT solve \( F_h \land (F_s \setminus \{ C_1, C_9 \}) \)
- UNSAT core \( K = \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ \begin{align*}
C_1 &= x_6 \lor x_2 \\
C_4 &= \neg x_1 \\
C_7 &= x_2 \lor x_4 \\
C_{10} &= \neg x_7 \lor x_5 \\
C_2 &= \neg x_6 \lor x_2 \\
C_5 &= \neg x_6 \lor x_8 \\
C_8 &= \neg x_4 \lor x_5 \\
C_{11} &= \neg x_5 \lor x_3 \\
C_3 &= \neg x_2 \lor x_1 \\
C_6 &= x_6 \lor \neg x_8 \\
C_9 &= x_7 \lor x_5 \\
C_{12} &= \neg x_3
\end{align*} \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \}, \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \} \]

- Update \( \mathcal{K} := \mathcal{K} \cup \{ K \} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \}, \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \} \]

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \}, \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \} \]

- Solve minimum-cost hitting set problem over \( \mathcal{K} \leadsto hs = \{ C_4, C_9 \} \)
MaxSAT by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]
\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]
\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]
\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

\[ \mathcal{K} := \{ \{ C_1, C_2, C_3, C_4 \}, \{ C_9, C_{10}, C_{11}, C_{12} \}, \{ C_3, C_4, C_7, C_8, C_{11}, C_{12} \} \} \]

- SAT solve \( F_h \land (F_s \setminus \{ C_4, C_9 \}) \)
MaxSAT by SAT and Hitting Set Computation: Example

\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*}

\[ \mathcal{K} := \{\{C_1, C_2, C_3, C_4\}, \{C_9, C_{10}, C_{11}, C_{12}\}, \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}\} \]

- SAT solve \( F_h \land (F_s \setminus \{C_4, C_9\}) \) \( \leadsto \) SATISFIABLE.
MaxSAT by SAT and Hitting Set Computation: Example

\[
\begin{align*}
C_1 &= x_6 \lor x_2 & C_2 &= \neg x_6 \lor x_2 & C_3 &= \neg x_2 \lor x_1 \\
C_4 &= \neg x_1 & C_5 &= \neg x_6 \lor x_8 & C_6 &= x_6 \lor \neg x_8 \\
C_7 &= x_2 \lor x_4 & C_8 &= \neg x_4 \lor x_5 & C_9 &= x_7 \lor x_5 \\
C_{10} &= \neg x_7 \lor x_5 & C_{11} &= \neg x_5 \lor x_3 & C_{12} &= \neg x_3
\end{align*}
\]

\[\mathcal{K} := \{\{C_1, C_2, C_3, C_4\}, \{C_9, C_{10}, C_{11}, C_{12}\}, \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}\}\]

- SAT solve $F_h \land (F_s \setminus \{C_4, C_9\}) \leadsto \text{SATISFIABLE.}$
  Optimal cost: 2 (cost of $hs$).
Optimizations

Solvers implementing the implicit hitting set approach include several optimizations, such as

- a *disjoint phase* for obtaining several cores before/between hitting set computations
- combinations of greedy and exact hitting sets computations
- ...

Some of these optimizations are *integral* for making the solvers competitive.

For more on some of the details, see

[Davies and Bacchus, 2011, 2013b,a]

[Saikko, Berg, and Järvisalo, 2016]
Implicit Hitting Set Approach to MaxSAT: Summary

- Effective on range of MaxSAT problems including large ones
- Superior to other methods when there are many distinct weights
- Usually superior to CPLEX
- On problems with no weights or very few weights can be outperformed by SAT-based approaches
Iterative Use of SAT Solvers for MaxSAT
Iterative Use of SAT Solvers (for MaxSAT)

- In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions.
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”

- Various modern CDCL SAT solvers implement an API for solving under assumption:
  - The input formula is read in only once.
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions.
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls.
  - The solver can keep its internal state from the previous solver call to the next.
    - Learned clauses
    - Heuristic scores

Järvisalo (U Helsinki)
Iterative Use of SAT Solvers (for MaxSAT)

- In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”

- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
Iterative Use of SAT Solvers (for MaxSAT)

- In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”

- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
Iterative Use of SAT Solvers (for MaxSAT)

- In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called assumptions
  - “Is the formula $F$ satisfiable under the assumption $x = 1$?”

- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - The solver can keep its internal state from the previous solver call to the next
    - Learned clauses
    - Heuristic scores
Iterative Use of SAT Solvers (for MaxSAT)

In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.

- Such forced partial assignments are called assumptions
- “Is the formula $F$ satisfiable under the assumption $x = 1$?”

Various modern CDCL SAT solvers implement an API for solving under assumption

- The input formula is read in only once
- The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
- The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
- The solver can keep its internal state from the previous solver call to the next
  - Learned clauses
  - Heuristic scores
Iterative Use of SAT Solvers (for MaxSAT)

In many application scenarios, including MaxSAT: it is beneficial to be able to make several SAT checks on the same input CNF formula under different forced partial assignments.

- Such forced partial assignments are called assumptions
- “Is the formula $F$ satisfiable under the assumption $x = 1$?”

Various modern CDCL SAT solvers implement an API for solving under assumption

- The input formula is read in only once
- The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
- The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
- The solver can keep its internal state from the previous solver call to the next
  - Learned clauses
  - Heuristic scores
Explaining Unsatisfiability

CDCL SAT solvers determine unsatisfiability when learning the empty clause

- By propagating a conflict at decision level 0

Explaining unsatisfiability under assumptions

- The reason for unsatisfiability can be traced back to assumptions that were necessary for propagating the conflict at level 0.
- Essentially:
  - Force the assumptions as the first “decisions”
  - When one of these decisions results in a conflict: trace the reason of the conflict back to the forced assumptions
Implementing MaxSAT Algorithms via Assumptions

SAT-based MaxSAT algorithms make use of the assumptions interface in SAT solvers

- Instrument each soft clause $C_i$ with a new “assumption” variable $a_i$
  - replace $C_i$ with $(C_i \lor a_i)$ for each soft clause $C_i$
- $a_i = 0$ switches $C_i$ “on”
- $a_i = 1$ switches $C_i$ “off”
Implementing MaxSAT Algorithms via Assumptions

SAT-based MaxSAT algorithms make use of the assumptions interface in SAT solvers

- Instrument each soft clause $C_i$ with a new “assumption” variable $a_i$
  - replace $C_i$ with $(C_i \lor a_i)$ for each soft clause $C_i$
- $a_i = 0$ switches $C_i$ “on”,
  - $a_i = 1$ switches $C_i$ “off”
- MaxSAT core: a subset of the assumptions variables $a_i$s
  - Heavily used in core-based MaxSAT algorithms
  - In the implicit hitting set approach:
    hitting sets over sets of assumption variables
  - Cost of including $a_i$ in a core (i.e., assigning $a_i = 1$):
    weight of the soft clause $C_i$

- Can state cardinality constraints directly over the assumption variables
  - Heavily used in MaxSAT algorithms employing cardinality constraints
Summary
MaxSAT

- Low-level constraint language: weighted Boolean combinations of binary variables
  - Gives tight control over how exactly to encode problem
- Exact optimization: provably optimal solutions
- MaxSAT solvers:
  - build on top of highly efficient SAT solver technology
  - various alternative approaches:
    - branch-and-bound, model-based, core-based, hybrids, . . .
  - standard WCNF input format
  - yearly MaxSAT solver evaluations

Success of MaxSAT

- Attractive alternative to other constrained optimization paradigms
- Number of applications increasing
- Solver technology improving rapidly
Further Topics

In addition to what we covered today:

MaxSAT is an active area of research, with recent work on

- preprocessing

  - How to simplify MaxSAT instances to make them easier for solver(s)?

- Parallel MaxSAT solving

  - How employ computing clusters to speed-up MaxSAT solving?

- Variants and generalization

  - MinSAT

  - Quantified MaxSAT
Further Topics

- instance decomposition/partitioning
  - [Martins, Manquinho, and Lynce, 2013]
  - [Neves, Martins, Janota, Lynce, and Manquinho, 2015]
- modelling high-level constraints
  - [Argelich, Cabiscol, Lynce, and Manyà, 2012]
  - [Zhu, Li, Manyà, and Argelich, 2012]
  - [Heras, Morgado, and Marques-Silva, 2015]
- understanding problem/core structure
  - [Li, Manyà, Mohamedou, and Planes, 2009]
  - [Bacchus and Narodytska, 2014]
  - [Li, Manyà, and Planes, 2006]
  - [Lin, Su, and Li, 2008]
  - [Li, Manyà, Mohamedou, and Planes, 2010]
  - [Li, Manyà, Mohamedou, and Planes, 2010]
  - [Heras, Morgado, and Marques-Silva, 2012]
- Lower/upper bounds
  - [Li, Manyà, Mohamedou, and Planes, 2010]
  - [Li, Manyà, Mohamedou, and Planes, 2010]
  - [Heras, Morgado, and Marques-Silva, 2012]
- symmetries
  - [Marques-Silva, Lynce, and Manquinho, 2008]
- ...
Further Reading and Links

**Surveys**

- Handbook chapter on **MaxSAT**: [Li and Manyà, 2009]
- Surveys on **MaxSAT** algorithms:
  - [Ansótegui, Bonet, and Levy, 2013a]
  - [Morgado, Heras, Liffiton, Planes, and Marques-Silva, 2013]

**MaxSAT Evaluation**

**Overview articles:**

- [Argelich, Li, Manyà, and Planes, 2008b]
- [Argelich, Li, Manyà, and Planes, 2011]

[http://maxsat.ia.udl.cat](http://maxsat.ia.udl.cat)
Thank you for your attention!


Bibliography VI


Bibliography VII


Bibliography VIII


António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided MaxSat with soft cardinality constraints. In O'Sullivan [2014], pages 564–573. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_41. URL http://dx.doi.org/10.1007/978-3-319-10428-7_41.


Zhu Zhu, Chu Min Li, Felip Manyà, and Josep Argelich. A new encoding from minsat into MaxSat. In Milano [2012], pages 455–463. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7_34. URL http://dx.doi.org/10.1007/978-3-642-33558-7_34.