MaxSat

Fahiem Bacchus

University of Toronto
Canada
Thanks also to Matti Järvisalo for many slides

February 12, 2016  AAAI-16, Phoenix, AZ, USA
SAT Solvers

SAT solving has been a great success.

From 100 variables, 200 constraints (early 90s) up to >10,000,000 vars. and >50,000,000 clauses. in 20 years.
Optimization

Most real-world problems involve an optimization component
Most real-world problems involve an optimization component

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.”

Benefits of provably optimal solutions:

- Resource savings
  - Money
  - Human resources
  - Time

- Accuracy

- Better approximations
  - by optimally solving reformulated problems
Importance of Exact Optimization

Giving Up?

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

Benefits of provably optimal solutions:

▶ Resource savings
  ▶ Money
  ▶ Human resources
  ▶ Time

▶ Accuracy
▶ Better approximations
  ▶ by optimally solving reformulated problems

Key Challenge: Scalability

Exactly solving instances of NP-hard optimization problems
Constrained Optimization Paradigms

**Mixed Integer-Linear Programming** (MIP, ILP)
- Constraint language:
  Conjunctions of linear inequalities
  \[ \sum_{i=1}^{k} c_i x_i \]
- Algorithms: e.g. Branch-and-cut w/Simplex

**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, Liffiton, and Marques-Silva, 2012]
[Zhang and Bacchus, 2012]
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
[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]
[Lynce and Marques-Silva, 2011]
[Guerra and Lynce, 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, 2016]
[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]
...
Progress in MaxSat Solver Performance

Comparing some of the best solvers from 2010–2014:
In 2014: 50% more instances solved than in 2010.

- On same computer, same set of benchmarks:
  Weighted Partial MaxSat encodings of “industrial”
MaxSat: Basic Definitions

- **Literal**: a boolean variable $x$ or $\neg x$.
- **Clause $C$**: a disjunction ($\lor$) of literals. e.g. $(x \lor y \lor \neg z)$
- **Truth assignment $\tau$**: a function from Boolean variables to $\{0, 1\}$.
  - $\tau(C) = 1$ if
    - $\tau(x) = 1$ for a literal $x \in C$,
    - $\tau(x) = 0$ for a literal $\neg x \in C$.

At least one literal of $C$ is made true by $\tau$. 

MaxSat: Basic Definitions

**MaxSat**

INPUT: a set of clauses $F$.
TASK: find $\tau$ s.t. $\sum_{C \in F} \tau(C)$ is maximized.

Find truth assignment that satisfies a maximum number of clauses

This is the standard definition, much studied in Theoretical Computer Science.

- Not convenient 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
- Clauses with finite weight are *soft*

Weighted Partial MaxSat

Hard clauses (partial) + weights on soft clauses (weighted)
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: 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: 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 $FP^{NP}$–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 $FP^{NP}$–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 NP=P).
Algorithms for Solving MaxSat
A variety of approaches MaxSat Algorithms

- Branch-and-bound
- Integer Programming (IP)
- SAT-Based Algorithms
  - Iterative
  - Core-guided
- Implicit hitting set algorithms (IP/SAT hybrid).
Examples of Recent MaxSAT Solvers by Category

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

**Model-based:**
- Q-MaxSAT
  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

- $UB$ is the cost of the best solution so far.
- $mincost(n)$ is the minimum cost achievable under node $n$.
- We can backtrack from $n$ when we know $mincost(n) \geq UB$ (no solution under $n$ is better).
- Our goal: calculate a lower bound $LB$ s.t. $mincost(n) \geq LB$.
- If $LB \geq UB$ then $mincost(n) \geq LB \geq UB$ and we can backtrack.
Lower Bounds

Common technique LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.
Lower Bounds

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

\[ \Phi = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots \]

Ignoring clause costs, \( \kappa = \{(x) \land (\neg x)\} \) is inconsistent.
Lower Bounds

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

\[ \Phi = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots \]

Ignoring clause costs, \( \kappa = \{(x) \land (\neg x)\} \) is inconsistent.

Let \( \kappa' = \{(\Box, 2) \land (\neg x, 1)\} \). Then \( \kappa' \) is MaxSat-equivalent to \( \kappa \): the cost of each truth assignment is preserved. (\( \Box \) is empty clause)
Lower Bounds

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

\[ \Phi = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots \]

Ignoring clause costs, \( \kappa = \{(x) \land (\neg x)\} \) is inconsistent.

Let \( \kappa' = \{(\Box, 2) \land (\neg x, 1)\} \).

Then \( \kappa' \) is MaxSat-equivalent to \( \kappa \): the cost of each truth assignment is preserved. (\( \Box \) is empty clause)

Let \( \Phi' = \Phi - \kappa \cup \kappa' \).

Then \( \Phi' \) is MaxSat-equivalent to \( \Phi \), and the cost of \( \Box \) has been incremented by 2

Cost of \( \Box \) must be incurred: it is an LB
Lower Bounds

1. Detect an inconsistent subset $\kappa$ (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 $\square$
   - e.g. $\kappa$ replaced by $\kappa' = \{ (\square, 2) \land (\neg x, 1) \}$
   - This replacement increases cost of $\square$ by 2.

3. Repeat 1 and 2 until no further increment to the LB is possible
Fast detection of some Cores

- Treat the soft clauses as if they were hard and then use **Unit Propagation** to detect conflicts.
- The conflict clause and the clauses that generated it form a core.
- This can find inconsistent sub-formulas quickly. But only limited set of inconsistent sub-formulas.
Transforming the Formula

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

» **MaxRes** generalizes such transformations to provide a sound and complete inference rule for MaxSat

[Larrosa and Heras, 2005]

[Bonet, Levy, and Manyà, 2007]
MaxRes

- **MaxRes** is a rule of inference that like ordinary resolution takes as input two clauses and produces new clauses.
- Unlike resolution **MaxRes** (a) removes the input clauses and (b) produces multiple new clauses.
MaxRes

\[
\text{MaxRes} \left[ (x \lor a_1 \lor \ldots \lor a_s \lor w_1), (\neg x \lor b_1 \lor \ldots \lor b_t, w_2) \right] = \\
(a_1 \lor \ldots \lor a_s \ldots b_1 \lor \ldots \lor b_t, \min(w_1, w_2)) \\
(x \lor a_1 \lor \ldots \lor a_s, w_1 - \min(w_1, w_2)) \\
(\neg x \lor b_1 \lor \ldots \lor b_t, w_2 - \min(w_1, w_2)) \\
(x \lor a_1 \lor \ldots \lor a_s \lor \neg(b_1 \lor \ldots \lor b_t), \min(w_1, w_2)) \\
(\neg x \lor \neg(a_1 \lor \ldots \lor a_s) \lor b_1 \lor \ldots \lor b_t, \min(w_1, w_2))
\]

Regular Resolvent
Cost Reduced Input
One will vanish
Compensation Clauses
Must be converted to Clauses

[Larrosa and Heras, 2005; Bonet, Levy, and Manyà, 2007]
By adding the “compensation” clauses MaxRes preserves the cost of every truth assignment.

Bonet et al. give a directly clausal version and a systematic way of using MaxRes to derive the empty clause (□, Opt) with weight Opt equal to the optimal cost.

[Bonet, Levy, and Manyà, 2007]
Other Lower Bounding Techniques

- Falsified soft learnt clauses and hitting sets over their proofs
  [Davies, Cho, and Bacchus, 2010]

- Clone is an approach that used a relaxation of the MaxSat formula.
  [Pipatsrisawat, Palyan, Chavira, Choi, and Darwiche, 2008]

- This relaxation was compiled into a d-DNNF.
- The relaxation provides a quick LB at each node.
- Other relaxations including minibuckets, or width-restricted BDDs might be applied.
  [Dechter and Rish, 2003]
  [Bergman, Ciré, van Hoeve, and Yunes, 2014]
Branch and Bound Methods—Summary

- Can be effective on small combinatorially hard problems, e.g., maxclique in a graph.
- Once the number of variables gets to 1,000 or more it is less effective: LB techniques become weak or too expensive.
Integer Programming (IP)
Solving MaxSat with an IP Solver

- Optimization problems have been studied for decades in the field of operations research (OR).
- In OR the most commonly used tool for solving optimization problems are state-of-the-art Integer Program Solvers, like IBM’s CPLEX.
- These solvers solve problems with linear constraints and objective function where some variables are integers.
- Modern IP solvers are extremely effective and successful, so it is natural to consider using this tool for MaxSat as well.
Blocking Variables (Relaxation Variables)

MaxSat solving often uses blocking variables to relax (block) soft clauses.

- To a soft clause \((x_1 \lor x_2 \lor \cdots \lor x_k)\) we add a new variable \(b\):

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

\(b\) does not appear anywhere else in the formula.

- If we make \(b\) true the soft clause is automatically satisfied (is relaxed/is blocked).
- If we make \(b\) false the clause becomes hard and must be satisfied.
Solving MaxSat with an IP Solver

- To every soft clause \( c_i \) add a new “blocking” variable \( b_i \).

\[
(x \lor \neg y \lor z \lor \neg w) \Rightarrow (b_1 \lor x \lor \neg y \lor z \lor \neg w)
\]

- Convert every augmented clause into a linear constraint:

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

- Each variable is integer in the range \([0 – 1]\).

- Finally add the objective function

\[
\text{minimize } \sum_{i} b_i \ast \text{cost}(w_i)
\]
Solving MaxSat with an IP Solver

- IP solvers use Branch and Cut to solve.
  - 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.
- Modern IP solvers use lots of other techniques.
- For standard optimization problems, like minimum hitting sets (set cover) they are extremely effective.
- But in for problems where there are many hard boolean constraints IP is not as effective.
SAT-Based
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

Basic Framework (UNSAT ⇒ SAT). We successively relax the MaxSat formula allowing more and more soft clauses to be falsified.

1. Verify that the set of hard clauses are SAT
   If UNSAT STOP. There are no MaxSat solutions!

2. Try to solve input MaxSat problem $\Phi$ without allowing any soft clause to be falsified (unrelaxed $\Phi$).

3. If SAT STOP, found a truth assignment satisfying all soft clauses.

4. Else Repeat until $\Phi$ is SAT
   4.1 Modify $\Phi$ so that more soft clauses can be falsified (relax $\Phi$ further).
      Must find minimal relaxation needed for SAT to find optimal solution
   4.2 Try to solve $\Phi$ again.
Iterative SAT solving (Linear Search)

Simplest (and least effective) linear search approach. (Unit clause weights).

1. Input MaxSat CNF $\Phi$
2. Add blocking variable $b_i$ to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If $\text{SAT}(\Phi \cup \text{CNF}(\sum b_i \leq k))$ return $k$
5. Else $k = k + 1$ and repeat 4.
Iterative SAT solving (Linear Search)

1. Input MaxSat CNF $\Phi$
2. Add blocking variable $b_i$ to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If $\text{SAT}(\Phi \cup \text{CNF}(\sum b_i \leq k))$
   return $k$
5. Else $k = k + 1$ and repeat 4.

- $\text{CNF}$ converts cardinality constraint to CNF. By allowing $k$ of the $b_i$’s to be true we “remove” $k$ soft clauses.
- $\text{SAT}$: Can remaining clauses be satisfied after removing any set of up to $k$ soft clauses (the SAT solver searches for which ones to remove).
- If $k$ yields UNSAT we try removing $k + 1$ soft clauses.
- If $k$ yields SAT, prior UNSAT for $k - 1$ proves that the satisfying assignment is optimal.
Iterating over $k$

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

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

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

3. SAT to UNSAT (used in Q-MaxSAT, can be effective on certain problems)
   3.1 Find a satisfying assignment $\pi$ of the hard clauses.
   3.2 Solve with $k = (\# \text{ of clauses falsified by } \pi) - 1$
   3.3 If SAT found better assignment. Reset $k$ and repeat 2.
   3.4 If UNSAT last assignment $\pi$ found is optimal.

This method finds a sequence of improved models—thus can give an approximate solution.
SAT-Based MaxSat using Cores

Core
Given an unsatisfiable CNF formula $\Phi$, a core of $\Phi$ is a subset of $\Phi$ that is itself unsatisfiable.

Cores for MaxSat
A subset of soft clauses of $\Phi$ that together with the hard clauses of $\Phi$ is unsatisfiable.
Cores from SAT Solvers

- Modern SAT solvers can return a core when they determine that the input is UNSAT.
- By removing the hard clauses from the core, we obtain a core for MaxSat
  \((\kappa, SAT?) = sat(\Phi)\)
  Sat solve \(\Phi\). Return Boolean SAT or UNSAT status: \(SAT?\)
  If UNSAT, return a core \(\kappa\) (set of soft clauses).
- Different methods are available for obtaining the core:
  1. Using assumptions.
  2. Outputting a clausal proof and then obtaining a core by trimming it.
Core-Based MaxSat Solving

- In the linear approach we add $CNF(\sum b_i \leq k)$ to the SAT solver.
- There is one $b_i$ for every soft clause in the theory. This cardinality constraint could be over 100,000s of variables: it is very loose.
- No information about which particular blocking variables to make true.
- This makes SAT solving inefficient: could have to explore many choices of subsets of $k$ soft clauses to remove.
- However, if we obtain a core we have a powerful constraint on which particular soft clauses need to be removed.
Constraint from Cores

- If $\kappa$ is a MaxSat core, then at least one if the soft clauses in it must be removed: no truth assignment satisfies every clause in $\kappa$ along with all of the hard clauses.
- Typically cores are much smaller than the set of all soft clauses.
MUS3

MUS3 is an simple MaxSat algorithm for exploiting cores

[Marques-Silva and Planes, 2007].

1. Input MaxSat CNF $\Phi$
2. $k = 0; BV = \{}$
3. $(\kappa, SAT?) = \text{sat}(\Phi)$
4. If $SAT?$ return $k$.
5. $k = k + 1$
6. Update $\Phi$:
   6.a For $c \in \kappa$ if $c$ has no blocking variable
      $c = c \cup \{b\}$ (new blocking variable $b$);
      $BV = BV \cup \{b\}$
   6.b Remove previous cardinality constraint.
   6.c Add $CNF(\sum_{b \in BV} b \leq k + 1)$
7. GOTO 3
1. Input MaxSat CNF $\Phi$
2. $k = 0; BV = \{\}$.
3. $(\kappa, SAT?) = \text{sat}(\Phi)$
4. If $SAT?$ return $k$.
5. $k = k + 1$
6. Update $\Phi$:
   6.a For $c \in \kappa$ if $c$ has no blocking variable
      $c = c \cup \{b\}$
      $BV = BV \cup \{b\}$
   6.b Remove previous cardinality constraint.
   6.c Add $CNF(\sum_{b \in BV} b \leq k + 1)$
7. GOTO 3

- Initially NO blocking variables!
- The cardinality constraint is always only over soft clauses that have participated in some core.
- The blocking variables in the cardinality constraint grows as more cores are discovered.
- On many problems however the cardinality constraint remains over a proper subset of the soft clauses.
1. By itself MUS3 is not effective.
2. Very effective when combined with an incremental construction of the cardinality constraint (so that each new constraint builds on the encoding of the previous constraint).
   
   [Martins, Joshi, Manquinho, and Lynce, 2014]

3. OpenWBO uses MUS3 with incremental cardinality constraints to achieve state-of-the-art performance on many problems.
Overlapping Cores

Say that the first two cores MUS3 finds are \( \{c_1, c_2\} \) and \( \{c_3, c_4\} \).

MUS3 would construct and use the cardinality constraint
\[
(b_1 + b_2 + b_3 + b_4 \leq 2).
\]
Where the \( b_i \) are the corresponding blocking variables.

But we actually know something stronger:
\[
(b_1 + b_2 \leq 1) \text{ and } (b_3 + b_4 \leq 1).
\]

It would be stronger to use a separate cardinality constraint for each core.
Overlapping Cores

- However, overlapping cores pose a problem!
- Each core is generated from an updated formula that has been relaxed to account for the previous cores.
- Say the first and second cores are
  1. \{\lnot a, 1\}, \{\lnot b, 1\}
  2. \{\lnot b, b, 1\}, \{\lnot c, 1\}, \{\lnot g, 1\}
- The original soft clause \lnot b, 1 participates in both cores!
- Core 2 is a core of the updated formula that includes\n  (\lnot a \lor b, 1), (\lnot b \lor b, 1) and (b + b \leq 1): one of a or b can be true.
- Core 2 is an unsatisfiable set of soft clauses even when we are allowed to set one of a or b to true.

Core 2 and Core 1 imply that we have

\[(a \land (b \lor c \lor g)) \lor (b \land (c \lor g))\]
Overlapping Cores

- For these cores the constraint
  \((b_a + b_b \leq 1) \land (b_b + b_c + b_g \leq 1)\) is too strong: E.g. 
  \(b \land c\) is a solution of

  \((a \land (b \lor c \lor g)) \lor (b \land (c \lor g))\)

  but is not a solution of the two cardinality constraints

  \((b_a + b_b \leq 1) \land (b_b + b_c + b_g \leq 1)\)

- Dealing with overlapping cores is a complicating issue for most core-based algorithms.
Fu-Malik

Fu-Malik deals with overlapping cores by adding a new blocking variables to the clauses in the core even when they already have a previous blocking variable. [Fu and Malik, 2006]

1. \( k = 0, \kappa = \{(\neg a), (\neg b)\} \)
2. Update these clauses to \( \{(\neg a, b_a), (\neg b, b_b)\} \).
3. Add \( CNF(b_a + b_b \leq 1) \)
4. \( k = 1, \kappa = \{(\neg b, b_b), (\neg c), (\neg g)\} \)
5. Update these clauses to \( \{(\neg b, b_b, b_b^2), (\neg c, b_c), (\neg g, b_g)\} \)
6. Add \( CNF(b_b^2 + b_c + b_g \leq 1) \)

But this allows \((\neg b, b_b, b_b^2)\) to be falsified in two different ways.
Another method for dealing with overlapping cores was developed by Ansótegui et al. [Ansótegui, Bonet, and Levy, 2013a]

- Only one blocking variable per soft clause.
- Group intersecting cores into disjoint covers. The cores might not be disjoint but the covers will be.
- Put a distinct at-most $\leq$ cardinality constraint over the soft clauses in a cover. Disjoint so this works.
- Keep an at-least $\geq$ constraint over the clauses in a core. Mainly to provide extra constraints for the SAT solver.
Recent advances in SAT-Based MaxSat solving come from approaches that add new variables to the formula. New variables always been used encoding the cardinality constraint. No attention was paid to the structure of these variables. The current top SAT-Based approaches EVA, MSCG-OLL, OpenWBO, WPM3 and maxino all exploit new variables. EVA, MSCG-OLL and WPM3 explicitly add new variables. OpenWBO and maxino more carefully structure the new variables arising from encoding the cardinality constraints.
When a new core $K = \{c_1, c_2, \ldots, c_p\}$ is found Eva uses the following technique to update the formula. [Narodytska and Bacchus, 2014]

1. For each $c_i \in K$ a new variable $b_i$ is made equivalent to $\neg b_i \equiv c_i$ 
   (hard clauses).
2. At least one clause in the core must be falsified: 
   $(b_1 \lor b_2 \lor \cdots \lor b_p)$
   (hard clause)
3. A new variable $d_i$ is made equivalent to sufixes of the core disjunction $d_i \equiv (b_{i+1} \lor \cdots \lor b_p)$
   (hard clauses)
4. $\{c_1, c_2, \ldots, c_p\}$ are removed.
5. $k = k + 1$
6. Soft clauses 
   $\{ (\neg b_1 \lor \neg d_1, 1), (\neg b_2 \lor \neg d_2, 1), \ldots, (\neg b_{p-1} \lor \neg d_{p-1}, 1) \}$ added.
Core = \{ c_1, c_2, c_3, c_4, c_5 \}
\neg b_1 \equiv c_1 \quad \neg b_2 \equiv c_2 \quad b_3 \equiv c_3 \quad b_4 \equiv c_4 \quad b_5 \equiv c_5
(b_1 \lor b_2 \lor b_3 \lor b_4 \lor b_5)
\neg d_1 \equiv \neg b_2 \land \neg b_3 \land \neg b_4 \land \neg b_5
\neg d_2 \equiv \neg b_3 \land \neg b_4 \land \neg b_5
\neg d_3 \equiv \neg b_4 \land \neg b_5
\neg d_4 \equiv \neg b_5
(\neg b_1 \lor \neg d_1, 1) \quad \text{soft clauses}
(\neg b_2 \lor \neg d_2, 1)
(\neg b_3 \lor \neg d_3, 1)
(\neg b_4 \lor \neg d_4, 1)
Only one of the soft clauses \( \{c_1, \ldots, c_5\} \) is made false (say \( c_3 \)):

1. \( \neg b_1, \neg b_2, b_3, \neg b_4, \neg b_5 \).
2. Most new soft clauses \( (\neg b_i \lor \neg d_i, 1) \) satisfied by \( \neg b_i \).
3. \( (\neg b_3, \neg d_3, 1) \)
   \[ \neg d_3 \equiv \neg b_4 \land \neg b_5 \]
   so \( \neg d_3 \) is true
   this soft clause is satisfied.

So no cost is incurred in the new formula. (new formula is relaxed)
E.g., if more than one soft clause is made false (say $c_2$ and $c_3$):

1. $\neg b_1, b_2, b_3, \neg b_4, \neg b_5$.
2. $(\neg b_1 \lor \neg d_1, 1), (\neg b_4 \lor \neg d_4, 1)$ are satisfied
3. $\neg d_2 \equiv \neg b_3 \land \neg b_4 \land \neg b_5$
   so $\neg d_2$ is false
   $(\neg b_2 \lor \neg d_2, 1)$ is falsified.
4. $\neg d_3 \equiv \neg b_4 \land \neg b_5$
   so $\neg d_3$ is true
   $(\neg b_3 \lor \neg d_3, 1)$ is satisfied

So if 2 of the $c_i$ are falsified one new soft clause is falsified incurring cost 1.
These new $d_i$ variables capture a disjunction of soft clauses (c.f. extended resolution).

If a future core involves ($\neg b_i \lor \neg d_i, 1$) EVA will introduce a new variable $x_i \equiv (\neg b_1 \lor \neg d_i)$.

And a new variable $\neg y_i \equiv \neg x_{i+1} \land \cdots \land \neg x_s$ capturing conjunctions of the $\neg x_i$.

This process can be repeated.

In this way variables can be introduced representing complex conditions.

These variables seem to help the SAT solver in finding new cores.

But a deeper understanding of this has not yet been developed.
Soft Cardinality Constraints

- MSCG-OLL and later WPM3 both introduce new variables equivalent to the cardinality constraints.
- \( d \equiv b_1 + b_2 + b_3 + b_4 + b_5 \leq 1. \)
- These new variables are used in new soft clauses that incur a cost if the cardinality constraint is violated.
- \((d, 1)\) is falsified if \( b_1 + b_2 + b_3 + b_4 + b_5 > 1. \)
- The \((d, 1)\) soft clauses can participate in new cores.
- Again these variables seem to help the SAT solver in finding new cores.

[Morgado, Dodaro, and Marques-Silva, 2014]
[Ansótegui, Didier, and Gabàs, 2015]
New Variables in Cardinality Constraints

- openWBO and maxino develop special methods for constructing the cardinality constraints associated with each core.
- They build them in such a way that each new cardinality constraint can share variables with the previous constraints.
- This tends to generate new variables expressing the sum of useful sets of soft constraints (soft constraints that appear together in more than one core).
- Again these variable seem to help the SAT solver

[Martins, Joshi, Manquinho, and Lynce, 2014]
[Alviano, Dodaro, and Ricca, 2015]
Dealing with Weighted Soft Clauses

- The methods presented can process a new core whenever all of the soft clauses have the same weight.
- If this weight is \( w \) we can set \( k = k + w \) rather than \( k = k + 1 \).
- So the common method used is **clause cloning**
  1. Let \( K \) be a core (set of soft clauses).
  2. Let \( w_{\text{min}} \) be the minimum weight soft clause in \( K \).
  3. We split each clause \((c, w) \in K\) into two clauses
     (1) \((c, w_{\text{min}})\) (2) \((c, w - w_{\text{min}})\).
  4. Keep all clauses (2) \((c, w - w_{\text{min}})\) as soft clauses (discard zero weight clauses)
  5. We let \( K \) be all clauses (1) \((c, w_{\text{min}})\)
  6. We process \( K \) as a new core (all clauses in \( K \) have the same weight)
Sat Based MaxSat

- Techniques are effective on large MaxSat problems, especially those with many hard clauses.
- The innovation is in obtaining more efficient ways to encode and solve the individual SAT decision problems that have to be solved.
- Some work done on understand the core structure and its impact on SAT solving efficiency [Bacchus and Narodytska, 2014], but more needed.
- The method of clause cloning for dealing with varying clause weights is not effective when there are many different weights.
- Many weighted problems in the MaxSat evaluation have very few different weights.
Implicit Hitting Set Algorithms for MaxSat

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

MaxSat
For any MaxSat instance $F$:
For any optimal hitting set $H$ of the set of UNSAT cores of $F$, $F \setminus H$ is satisfiable
And any satisfying assignment of $F \setminus H$ is an optimal solution
Hitting Sets and UNSAT Cores

MaxSat
For any MaxSat instance $F$:
For any optimal hitting set $H$ of the set of UNSAT cores of $F$, $F \setminus H$ is satisfiable
And any satisfying assignment of $F \setminus H$ is an optimal solution

Hitting Sets
Given a set $S$ of elements,
A set $H$ is a hitting set of $S$ is $H \cap S \neq \emptyset$ for all $S \in S$.
A hitting set $H$ is optimal if for every hitting set $H'$ of $S$
$\sum_{c \in H} \text{wt}(c) \leq \sum_{c \in H'} \text{wt}(c)$.
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 cores 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.
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

$$\min \sum_{c \in \bigcup \mathcal{K}} wt(c) \cdot b_c$$

subject to $\sum_{c \in K} b_c \geq 1 \quad \forall K \in \mathcal{K}$
Implicit Hitting Set Approach to MaxSat

“Best out of both worlds”
Combining 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
  - So the SAT instances are easier (no cardinality constraints).

- IP solvers are at optimization
  - Instead of directly solving the input MaxSAT instance:
    solve a sequence of (simpler) hitting set problems over the cores
Solving MaxSat by SAT and Hitting Set Computations

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

```
F_h, F_s
hs := ∅
K := ∅

K := K ∪ {K}
wt

F_h ∧ (F_s \ hs)
unsat

sat
hs of K

UNSAT core extraction

Min-cost Hitting Set

Optimal solution found

IP solver
\[
\min \sum_{c \in K} \text{wt}(c) \cdot b_c \\
\sum_{c \in K} b_c \geq 1 \ \forall K \in K
\]
```
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
   - $F_h, F_s$
   - $hs := \emptyset$
   - $\mathcal{K} := \emptyset$

**SAT solver**

$F_h \land (F_s \setminus hs)$

**UNSAT core extraction**

**Min-cost Hitting Set**

**IP solver**

\[
\begin{align*}
\text{min} & \sum_{c \in \mathcal{K}} \text{wt}(c) \cdot b_c \\
\sum_{c \in \mathcal{K}} b_c & \geq 1 \quad \forall K \in \mathcal{K}
\end{align*}
\]

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

2. UNSAT core

**SAT solver**
$F_h \land (F_s \setminus hs)$

**UNSAT core extraction**
$F_h, F_s$
$hs := \emptyset$
$\mathcal{K} := \emptyset$

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

Unsat

**Min-cost Hitting Set**
$hs$ of $\mathcal{K}$

$\mathcal{K} : unwt(k)$

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

Optimal solution found

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

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

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
$\min \sum_{c \in \cup \mathcal{K}} wt(c) \cdot b_c$
$\sum_{c \in K} b_c \geq 1 \ \forall K \in \mathcal{K}$

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

4. Min-cost HS of $\mathcal{K}$

SAT solver

$F_h \land (F_s \setminus hs)$

UNSAT core extraction

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

Min-cost Hitting Set

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

$wt$

IP solver

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

unsat

sat

hs of $\mathcal{K}$

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

5. UNSAT core

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

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

\[
\text{unsat}
\]

\[
\text{sat}
\]

**UnSAT core extraction**

**Min-cost Hitting Set**

\[
\text{IP solver}
\min \sum_{c \in \mathcal{K}} \text{wt}(c) \cdot b_c
\]

\[
\sum_{c \in \mathcal{K}} b_c \geq 1 \forall K \in \mathcal{K}
\]

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

iterate until “sat”

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

**UNSAT core extraction**

- SAT solver
- $F_h \land (F_s \setminus hs)$

**Min-cost Hitting Set**

- $\mathcal{K} := \mathcal{K} \cup \{K\}$
- $wt$
- $hs$ of $\mathcal{K}$

- IP solver
- $\min \sum_{c \in \cup \mathcal{K}} wt(c) \cdot b_c$
- $\sum_{c \in K} b_c \geq 1 \forall K \in \mathcal{K}$

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

iterate until “sat”

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

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

\[
\text{wt} 
\]

\[
\text{unsat} 
\]

\[
\text{sat} 
\]

\[
\text{hs of } \mathcal{K} 
\]

\[
\text{Optimal solution found} 
\]

SAT solver

\[
F_h \land (F_s \setminus hs) 
\]

UNSAT core extraction

\[
\text{IP solver} 
\]

\[
\min \sum_{c \in \mathcal{K}} \text{wt}(c) \cdot b_c \\
\sum_{c \in K} b_c \geq 1 \quad \forall K \in \mathcal{K} 
\]
Intuition: After optimally hitting all cores of $F_h \land F_s$ by $hs$: any solution to $F_h \land (F_s \setminus hs)$ is guaranteed to be optimal.

iterate until “sat”

**SAT solver**

$F_h, F_s$

$hs := \emptyset$

$\mathcal{K} := \emptyset$

**UNSAT core extraction**

$F_h \land (F_s \setminus hs)$

**Min-cost Hitting Set**

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

$wt$

$hs$ of $\mathcal{K}$

**IP solver**

min $\sum_{c \in \mathcal{K}} wt(c) \cdot b_c$

$\sum_{c \in K} b_c \geq 1 \ \forall K \in \mathcal{K}$

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

Many **improvements** have been made to speed up this basic algorithm.

---

**iterate until “sat”**

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

**SAT solver**

\[
F_h \land (F_s \setminus hs)
\]

**UNSAT core extraction**

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

**unsat**

**IP solver**

\[
\min \sum_{c \in \mathcal{K}} \text{wt}(c) \cdot b_c \\
\sum_{c \in K} b_c \geq 1 \ \forall K \in \mathcal{K}
\]

\[
\text{Optimal solution found}
\]

**sat**

**Min-cost Hitting Set**

hs of \(\mathcal{K}\)
MaxSat by SAT and Hitting Set Computation: Example

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

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

\[ \mathcal{K} := \emptyset \]
MaxSat by SAT and Hitting Set Computation: Example

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

\[ K := \emptyset \]

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

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

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

- 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 \( \mathcal{K} \)
\[ \leadsto hs = \{ C_1 \} \]
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 \( F_h \land (F_s \setminus \{ C_1 \}) \)
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

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

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

- 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

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

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

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

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

\( K := \{\{C_1, C_2, C_3, C_4\}, \{C_9, C_{10}, C_{11}, C_{12}\}, \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}\} \)

- Update \( K := K \cup \{K\} \)
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

\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*}

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

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

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

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

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

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

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

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

- 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.
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 application increasing
- Solver technology improving rapidly
Further Topics

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

- preprocessing
  - How to simplify MaxSat instances to make the 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]

- **Lower/upper bounds**
  - [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]

- **symmetries**
  - [Marques-Silva, Lynce, and Manquinho, 2008]

- ...


Bibliography II


Bibliography V


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.