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

- MaxSAT
- Complexity
- Use in practice

Overview of algorithmic approaches to MaxSAT

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

Use of SAT solvers for 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?

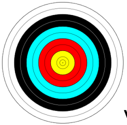
"The problem is NP-hard, so let's develop heuristics / approximation algorithms."

No!

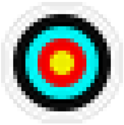
Benefits of provably optimal solutions:

- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

$$$



vs



Key Challenge: Scalability

Exactly solving instances of *NP-hard* optimization problems

Importance of Exact Optimization

Giving Up?

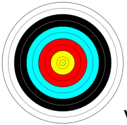
"The problem is NP-hard, so let's develop heuristics / approximation algorithms."

$$$

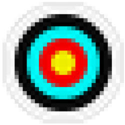
No!

Benefits of provably optimal solutions:

- Resource savings
  - Money, human resources, time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations



VS



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 $\sum_{i=1}^{k} c_i x_i \leq b$
- Algorithms: e.g. Branch-and-cut w/Simplex

Normal form: integer domain variables x_i, constants c_i, a_i^j, b_j

$$
\begin{array}{ll}
\textsc{Minimize} & \displaystyle\sum_{i=1}^{k} c_i x_i \\
\textsc{subject to} & \displaystyle\sum_{i=1}^{k} a_i^1 x_i \leq b_1 \\
& \ldots \\
& \displaystyle\sum_{i=1}^{k} a_i^m x_i \leq b_m
\end{array}
$$

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	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016a]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]

...

MaxSAT Applications

Central to the increasing success: Advances in MaxSAT solver technology

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016a]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]

...

Basic Concepts

MaxSAT: Basic Definitions

MaxSAT

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

TASK: find τ 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
    $\rightsquigarrow$ Existence of solutions not guaranteed
- Clauses with finite weight are *soft*

Weighted Partial MaxSAT

Hard clauses (partial) + weights on soft clauses (weighted)

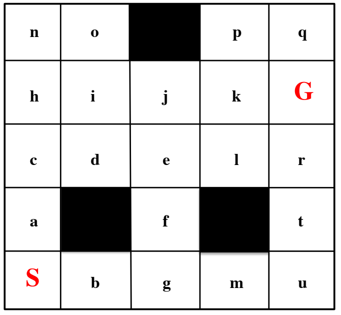
Terminology

- 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).



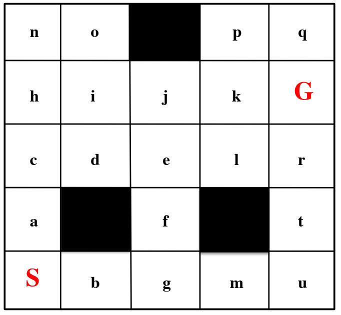
Note: best solved with state-space search

Here: to illustrate MAXSAT encodings

Example: Encoding shortest paths

Shortest Path

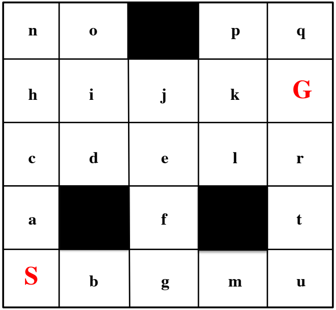
Find shortest path in a grid with horizontal/vertical moves.
Travel from S to G without entering blocked squares (black).



Note: best solved with state-space search

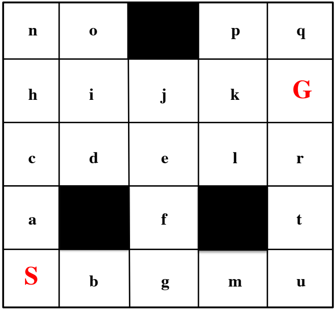
Here: to illustrate MAXSAT encodings

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: $(\neg a), (\neg b), \ldots, (\neg u)$ *"would prefer not to visit"*

MaxSAT: Example



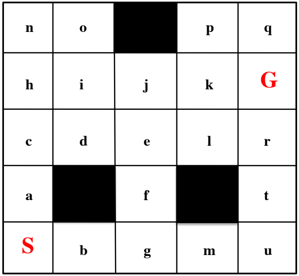
- Boolean variables: one for each unblocked grid square $\{S, G, a, b, \ldots, u\}$: true *iff path visits this square.*
- Constraints:
  - The S and G squares must be visited:
    In CNF: unit hard clauses (S) and (G).
  - A soft clause of weight 1 for all other squares:
    In CNF: $(\neg a)$, $(\neg b)$, ..., $(\neg u)$ "*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

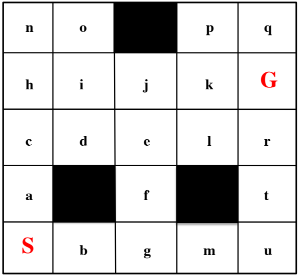


MaxSAT: Example

- 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



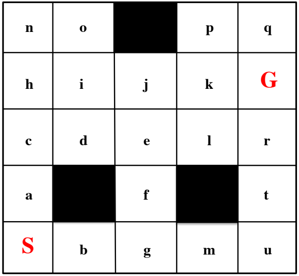
MaxSAT: Example

Constraint 1:

S and G must have exactly one visited neighbour.

- For S: $a + b = 1$
  - In CNF: $(a \vee b), (\neg a \vee \neg b)$
- For G: $k + q + r = 1$
  - "At least one" in CNF : $(k \vee q \vee r)$
  - "At most one" in CNF: $(\neg k \vee \neg q), (\neg k \vee \neg r), (\neg q \vee \neg r)$

    disallow pairwise



MaxSAT: Example

Constraint 1:

S and G must have exactly one visited neighbour.

- For S: $a + b = 1$
  - In CNF: $(a \vee b), (\neg a \vee \neg b)$
- For G: $k + q + r = 1$
  - "At least one" in CNF : $(k \vee q \vee r)$
  - "At most one" in CNF: $(\neg k \vee \neg q), (\neg k \vee \neg r), (\neg q \vee \neg r)$

    disallow pairwise

n	o		p	q
h	i	j	k	G
c	d	e	l	r
a		f		t
S	b	g	m	u

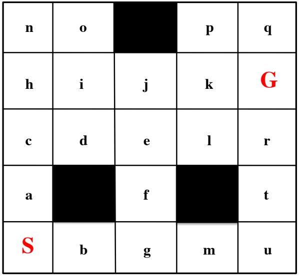
MaxSAT: Example

Constraint 1:

S and G must have exactly one visited neighbour.

- For S: $a + b = 1$
  - In CNF: $(a \vee b), (\neg a \vee \neg b)$
- For G: $k + q + r = 1$
  - "At least one" in CNF : $(k \vee q \vee r)$
  - "At most one" in CNF: $(\neg k \vee \neg q), (\neg k \vee \neg r), (\neg q \vee \neg r)$

disallow pairwise



MaxSAT: Example

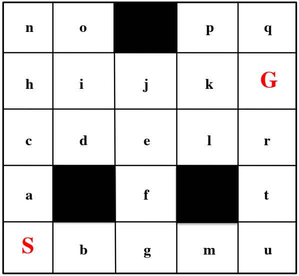
Constraint 2:

Other visited squares must have exactly two visited neighbours

- For example, for square e: $e \rightarrow (d + j + l + f = 2)$
  - 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



MaxSAT: Example

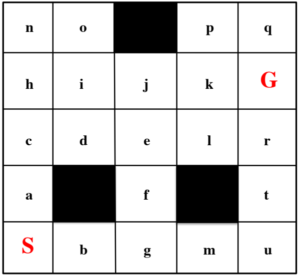
Constraint 2:

Other visited squares must have exactly two visited neighbours

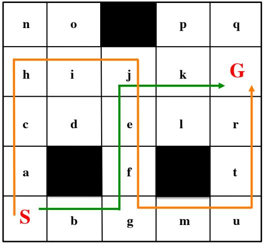
- For example, for square e: $e \rightarrow (d + j + l + f = 2)$
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

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



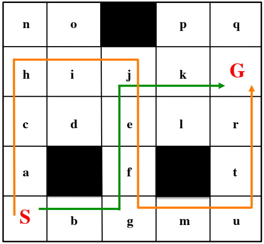
MaxSAT: Example



Properties of the encoding

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

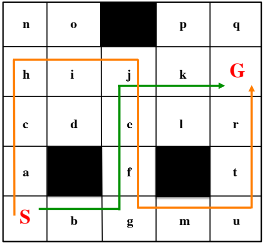
MaxSAT: Example



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: 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 pas 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

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 $\mathcal{C}$ with associated weight $W_{\mathcal{C}}$.

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

Softening $\text{CNF}(\mathcal{C})$ 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 $W_{\mathcal{C}}$.

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 $\mathcal{C}$ with associated weight $W_{\mathcal{C}}$.

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

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

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

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 $\mathcal{C}$ with associated weight $W_{\mathcal{C}}$.

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

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

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

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 $\mathrm{FP}^{\mathrm{NP}}$–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 $\mathrm{FP}^{\mathrm{NP}}$–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).

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–complete

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

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 $\mathrm{FP}^{\mathrm{NP}}$–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 $\mathrm{FP}^{\mathrm{NP}}$–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).

Practical MaxSAT Solving

Standard Solver Input Format: DIMACS WCNF

- Variables indexed from 1 to n
- Negation: -
  - -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
  - $(-x_3 \vee x_1 \vee \neg x_{45})$, weight 2:
    `2 -3 1 -45 0`
- Clause is hard if weight == `top`

Example:

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

MaxSAT Evaluations

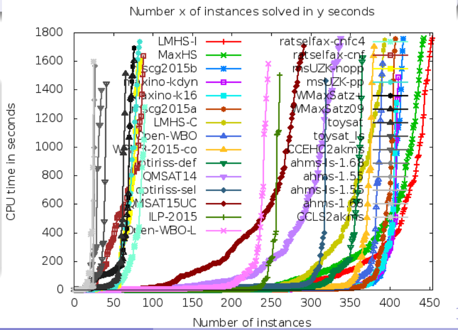
Objectives

- 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

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

Internally rely especially on CDCL SAT solvers

for proving unsatisfiability of subsets of clauses

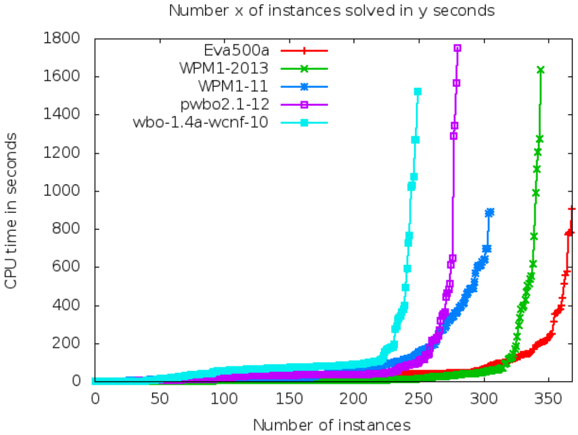
Push-Button Solver Technology

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
...
o 10548793370
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 287486453
c Total time: 1.30 s
c Nb SAT calls: 4
c Nb UNSAT calls: 841
s OPTIMUM FOUND
v 1 -2 3 4 5 6 7 8 -9 10 11 12 13 14 15 16 ...
... -18167 -18168 -18169 -18170

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__
- MaxSatz
  http://home.mis.u-picardie.fr/~cli/EnglishPage.html
- MSCG http://sat.inesc-id.pt/~aign/soft/
- WPM3 http://web.udl.es/usuaris/q4374304/#software
- QMaxSAT https://sites.google.com/site/qmaxsat/
- ...

Algorithms for MaxSAT Solving

A Variety of Approaches

Branch and bound:

- MaxSatz http://home.mis.u-picardie.fr/~cli/EnglishPage.html
- 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:

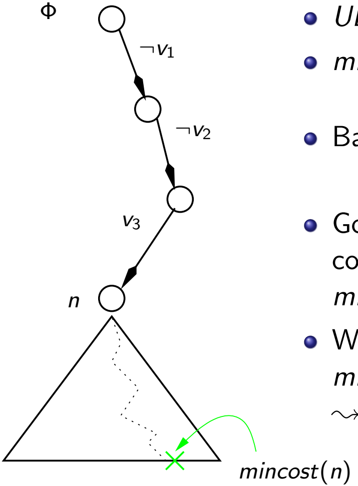
- Eva http://www.maxsat.udl.cat/14/solvers/eva500a__
- MSCG http://sat.inesc-id.pt/~aign/soft/
- OpenWBO http://sat.inesc-id.pt/open-wbo/
- WPM http://web.udl.es/usuaris/q4374304/#software
- maxino http://alviano.net/software/maxino/

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 $LB \geq UB$:
  $mincost(n) \geq LB \geq UB$
  $\rightsquigarrow$ 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 MAXSAT solvers:

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

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

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

- Then κ' is MaxSAT-*equivalent to* κ:
  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. $F = ... \wedge (x, 2) ... \wedge (\neg x, 3) ...$
Ignoring clause costs, $\kappa = \{(x), (\neg x)\}$ is unsatisfiable.

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

- Then κ' is MaxSAT-*equivalent to* κ:
  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 κ of clauses (aka core) of the current formula
   - e.g. $\kappa = \{(x, 2) \wedge (\neg x, 3)\}$
2. Apply sound transformation to the clauses in κ that result in an increment to the cost of the empty clause $\emptyset$
   - e.g. κ replaced by $\kappa' = \{(\emptyset, 2) \wedge (\neg x, 1)\}$
   - 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 $(x_1 \vee x_2 \vee \cdots \vee x_k)$:
  add a **new** variable r to obtain

$$(r \vee x_1 \vee x_2 \vee \cdots \vee x_k)$$

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.

$$(x \lor \neg y \lor z \lor \neg w) \rightsquigarrow (r_i \lor x \lor \neg y \lor z \lor \neg w)$$

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 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 $(C_1 \vee r_1) \wedge (C_2 \vee r_2) \wedge \cdots \wedge (C_n \vee r_n) \wedge$ $(\sum_{i=1}^{n} r_i \leq k)$
- Iterate over $k \in \{1, \ldots, n\}$ to find the optimal k
  - ...and an optimal solution.
  - ...*proving* that no solutions of cost $< k$ exist.

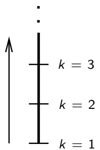
Iterative Search

Basic approach:

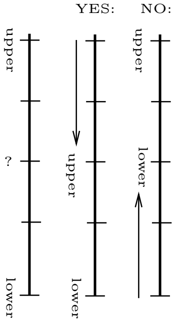
- To check whether F has a solution of cost $\leq k$:
  - SAT solve $(C_1 \vee r_1) \wedge (C_2 \vee r_2) \wedge \cdots \wedge (C_n \vee r_n) \wedge \underline{(\sum_{i=1}^{n} r_i \leq k)}$
- Iterate over $k \in \{1, \ldots, n\}$ to find the optimal k
  - ...and an optimal solution.
  - ...*proving* 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)
   - $UB =$ # 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 π 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 π 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 $F_h \wedge F_s'$ 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 $F_h \wedge F_s'$ 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 $F_h \wedge F_s'$ 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 $F_h \wedge F_s'$ 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 $F_h \wedge F_s'$ 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 $F_h \wedge F_s'$ 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)

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)

Fu-Malik: Example

(On an unweighted formula)

Example by Marques-Silva

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

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)

Example by Marques-Silva

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}
$$

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) Example by Marques-Silva

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \vee r_1 \\
C_4 = \neg x_1 \vee r_2 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 \vee r_3 & C_8 = \neg x_4 \vee x_5 \vee r_4 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 \vee r_5 & C_{12} = \neg x_3 \vee r_6
\end{array}
$$

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) Example by Marques-Silva

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \vee r_1 \\
C_4 = \neg x_1 \vee r_2 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 \vee r_3 & C_8 = \neg x_4 \vee x_5 \vee r_4 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 \vee r_5 & C_{12} = \neg x_3 \vee r_6
\end{array}
$$

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)

Example by Marques-Silva

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \vee r_1 \\
C_4 = \neg x_1 \vee r_2 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 \vee r_3 & C_8 = \neg x_4 \vee x_5 \vee r_4 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 \vee r_5 & C_{12} = \neg x_3 \vee r_6
\end{array}
$$

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)

Example by Marques-Silva

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

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) Example by Marques-Silva

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

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) Example by Marques-Silva

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

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)

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 kth iteration
- Relaxed soft clauses *become hard*

Core-Guided MaxSAT Algorithms: MSU3

(On an unweighted formula)

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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)

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}
$$

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)

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

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)

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

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)

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

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)

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

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)

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

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)

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

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)

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 $\leq$ cardinality constraints over the soft clauses in a cover
  - An at-least $\geq$ 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

Clause Cloning

Methor used to deal with varying weights

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

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

1. Split each clause $(c, w) \in K$ into two clauses:
   (1) $(c, w_{\min})$ and (2) $(c, w - w_{\min})$.
2. Keep all clauses (2) $(c, w - w_{\min})$ as soft clauses
   (discard zero weight clauses)
3. Let K be all clauses (1) $(c, w_{\min})$
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 understand 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 $\mathcal{S}$ of sets of elements,
A set H is a *hitting set* of $\mathcal{S}$ if $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.

A hitting set H is *optimal* if no $H' \subset \bigcup \mathcal{S}$ with $|H'| < |H|$ is a hitting set of $\mathcal{S}$.

- Note: Under weight function $c : S \to \mathbb{R}^+$, $c(H') < c(H)$ 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 τ to F such that τ satisfies exactly the clauses $F \setminus H$.

Hitting Sets and UNSAT Cores

Hitting Sets

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

A hitting set H is *optimal* if no $H' \subset \bigcup \mathcal{S}$ with $|H'| < |H|$ is a hitting set of $\mathcal{S}$.

- Note: Under weight function $c : S \to \mathbb{R}^+$, $c(H') < c(H)$ 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 τ to F such that τ satisfies exactly the clauses $F \setminus H$.

Hitting Sets and UNSAT Cores

Hitting Sets

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

A hitting set H is *optimal* if no $H' \subset \bigcup \mathcal{S}$ with $|H'| < |H|$ is a hitting set of $\mathcal{S}$.

- Note: Under weight function $c : S \to \mathbb{R}^+$, $c(H') < c(H)$ 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 τ to F such that τ 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

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

$$\text{subject to} \quad \sum_{C \in K} r_C \geq 1 \quad \forall K \in \mathcal{K}$$

- $r_C = 1$ iff clause C in the hitting set
- Weight function c: works also for weighted MaxSAT

Implicit Hitting Set Approach to 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

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

$$\text{subject to} \quad \sum_{C \in K} r_C \geq 1 \quad \forall K \in \mathcal{K}$$

- $r_C = 1$ iff clause C in the hitting set
- Weight function c: works also for weighted MaxSAT

Implicit Hitting Set Approach to MaxSAT

Intuition: combine the main strengths of SAT and IP solvers

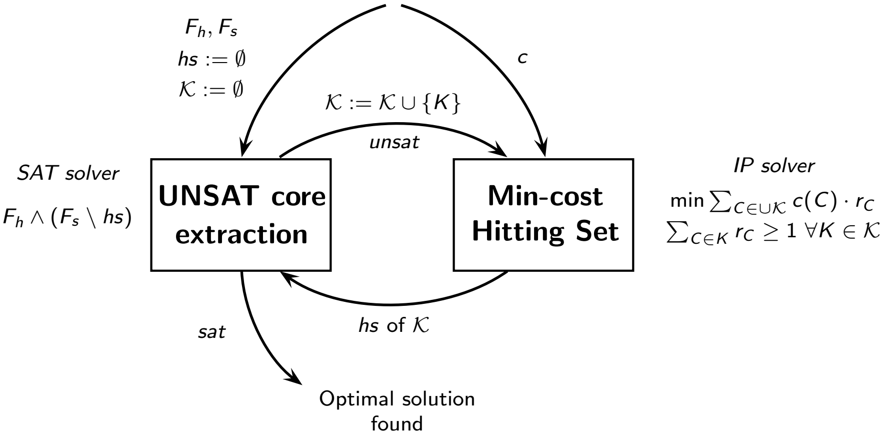
- SAT solvers are very good at proving unsatisfiability
  - Provide explanations for unsatisfiability in terms of cores
  - Instead of adding clauses to / modifying the input MaxSAT instance: each SAT solver call made on a *subset* of the clauses in the instance
- IP solvers at optimization
  - Instead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

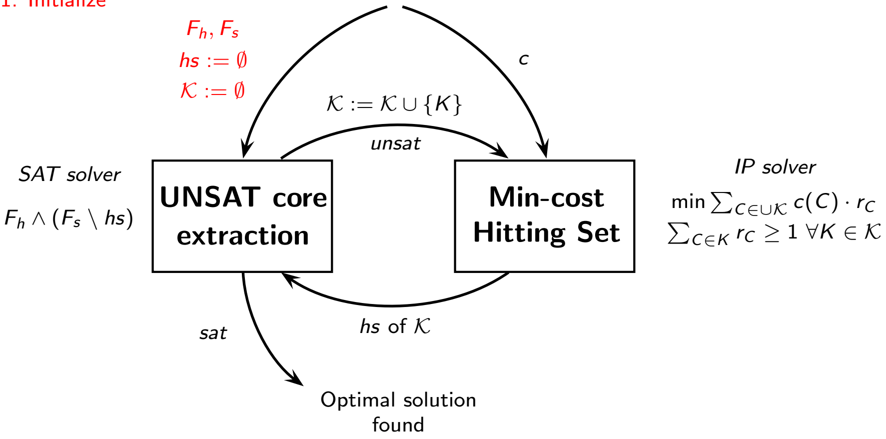
Input:
hard clauses F_h, soft clauses F_s, weight function $c : F_s \mapsto \mathbb{R}^+$



Solving MaxSAT by SAT and Hitting Set Computations

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

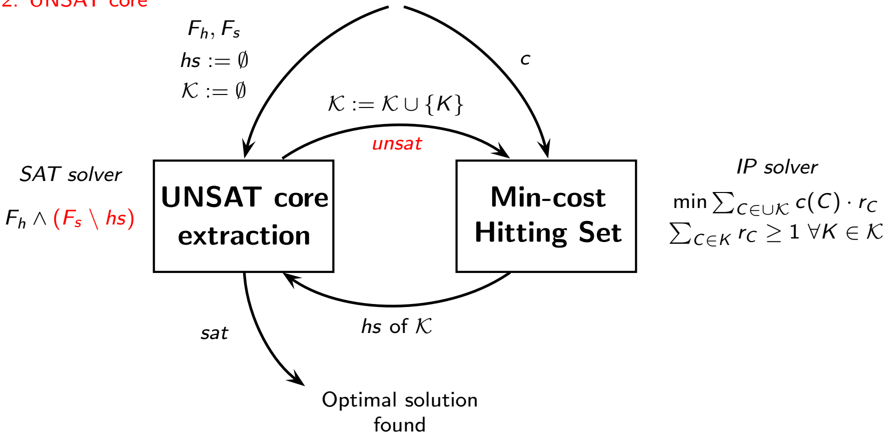
1. Initialize



Solving MaxSAT by SAT and Hitting Set Computations

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

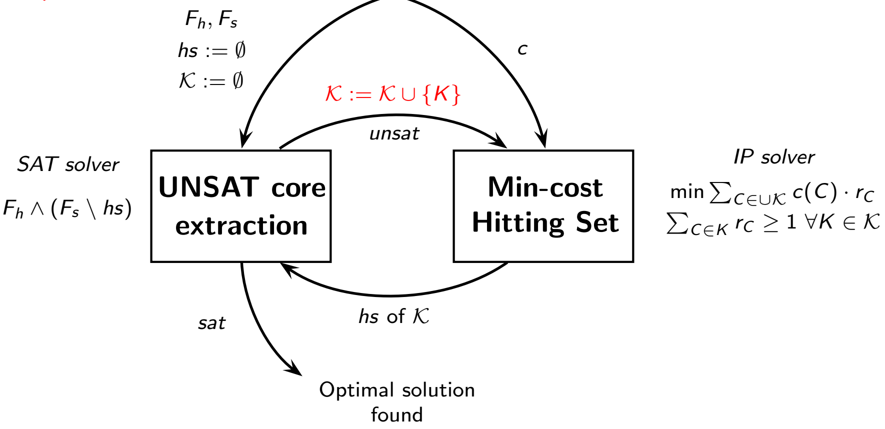
2. UNSAT core



Solving MAXSAT by SAT and Hitting Set Computations

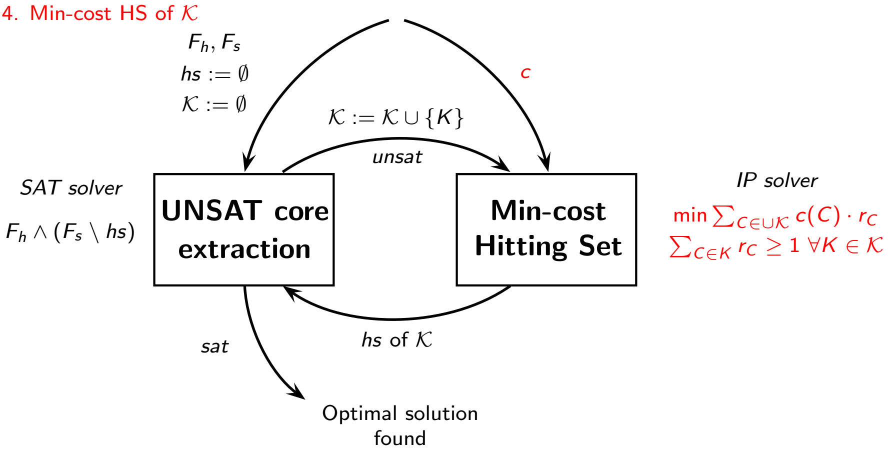
Input:
hard clauses F_h, soft clauses F_s, weight function $c \; : \; F_s \mapsto \mathbb{R}^+$

3. Update core set



Solving MaxSAT by SAT and Hitting Set Computations

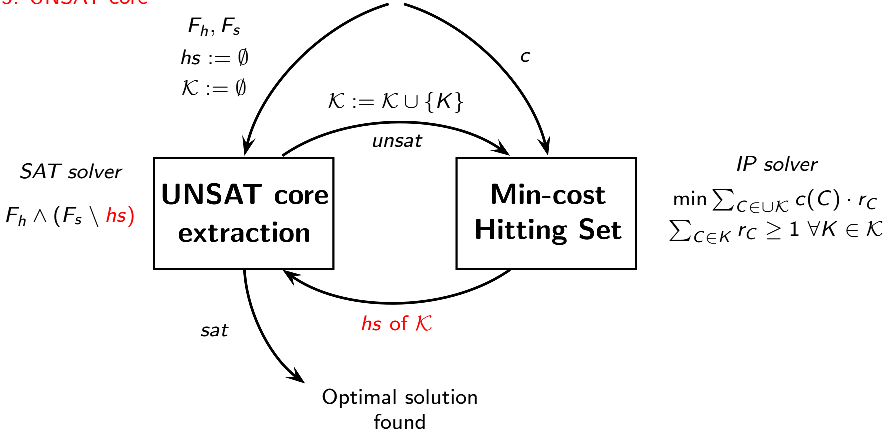
Input:
hard clauses F_h, soft clauses F_s, weight function $c : F_s \mapsto \mathbb{R}^+$



Solving MaxSAT by SAT and Hitting Set Computations

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

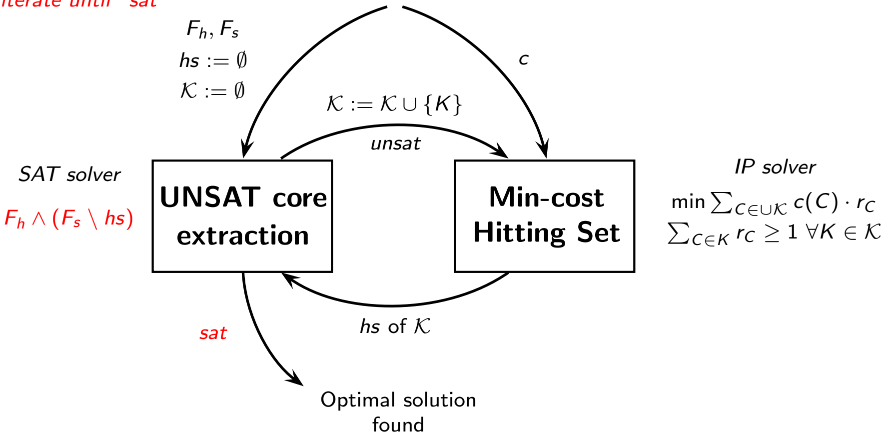
5. UNSAT core



Solving MaxSAT by SAT and Hitting Set Computations

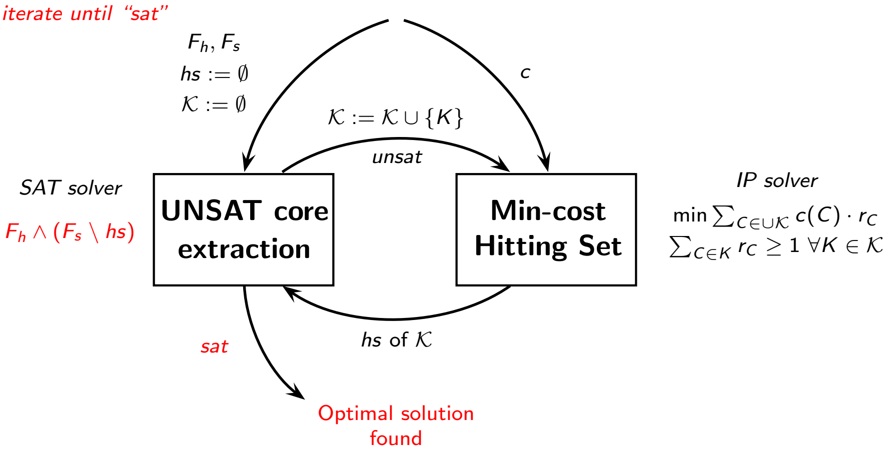
Input:
hard clauses F_h, soft clauses F_s, weight function $c : F_s \mapsto \mathbb{R}^+$

iterate until "sat"



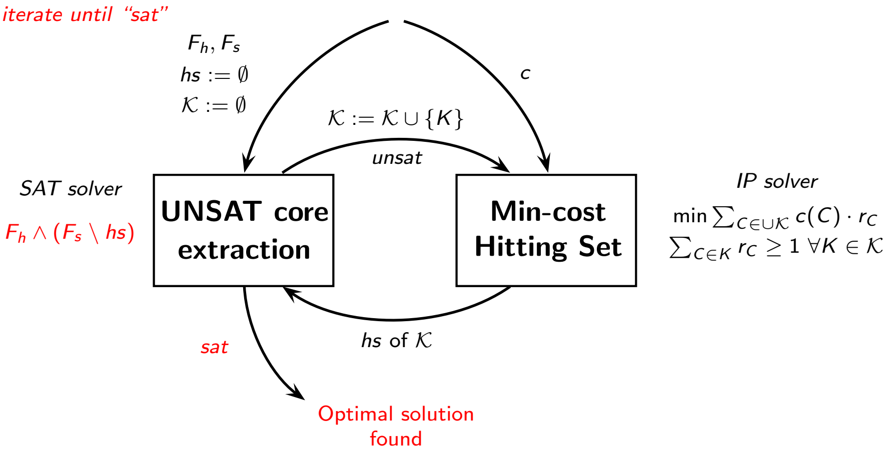
Solving MaxSAT by SAT and Hitting Set Computations

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



Solving MaxSAT by SAT and Hitting Set Computations

Intuition: After optimally hitting *all* cores of $F_h \wedge F_s$ by hs:
any solution to $F_h \wedge (F_s \setminus hs)$ *is guaranteed to be optimal.*



MaxSAT by SAT and Hitting Set Computation: Example

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

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \land (F_s \setminus \emptyset)$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}$$

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

- SAT solve $F_h \wedge (F_s \setminus \emptyset) \rightsquigarrow$ UNSAT core $K = \{C_1, C_2, C_3, C_4\}$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

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

MaxSAT by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over $\mathcal{K}$

MaxSAT by SAT and Hitting Set Computation: Example

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}
$$

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

- Solve minimum-cost hitting set problem over $\mathcal{K} \rightsquigarrow hs = \{C_1\}$

MaxSAT by SAT and Hitting Set Computation: Example

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}
$$

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

- SAT solve $F_h \wedge (F_s \setminus \{C_1\})$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- SAT solve $F_h \wedge (F_s \setminus \{C_1\}) \rightsquigarrow$ UNSAT core $K = \{C_9, C_{10}, C_{11}, C_{12}\}$

MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- Solve minimum-cost hitting set problem over $\mathcal{K}$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- Solve minimum-cost hitting set problem over $\mathcal{K} \rightsquigarrow hs = \{C_1, C_9\}$

MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \land (F_s \setminus \{C_1, C_9\})$

MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \land (F_s \setminus \{C_1, C_9\})$
  $\rightsquigarrow$ UNSAT core $K = \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

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

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- Solve minimum-cost hitting set problem over $\mathcal{K}$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- Solve minimum-cost hitting set problem over $\mathcal{K} \rightsquigarrow hs = \{C_4, C_9\}$

$\textsc{MaxSAT}$ by SAT and Hitting Set Computation: Example

$$
\begin{array}{ccc}
C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\
C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\
C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\
C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3
\end{array}
$$

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

- SAT solve $F_h \wedge (F_s \setminus \{C_4, C_9\})$

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- SAT solve $F_h \wedge (F_s \setminus \{C_4, C_9\}) \rightsquigarrow$ SATISFIABLE.

MaxSAT by SAT and Hitting Set Computation: Example

$$\begin{array}{ccc} C_1 = x_6 \vee x_2 & C_2 = \neg x_6 \vee x_2 & C_3 = \neg x_2 \vee x_1 \\ C_4 = \neg x_1 & C_5 = \neg x_6 \vee x_8 & C_6 = x_6 \vee \neg x_8 \\ C_7 = x_2 \vee x_4 & C_8 = \neg x_4 \vee x_5 & C_9 = x_7 \vee x_5 \\ C_{10} = \neg x_7 \vee x_5 & C_{11} = \neg x_5 \vee x_3 & C_{12} = \neg x_3 \end{array}$$

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

- SAT solve $F_h \wedge (F_s \setminus \{C_4, C_9\}) \rightsquigarrow$ 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

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
  $\rightsquigarrow$ 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
  $\rightsquigarrow$ 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_is
  - 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 [Argelich, Li, and Manyà, 2008a] [Belov, Morgado, and Marques-Silva, 2013] [Berg, Saikko, and Järvisalo, 2015b] [Berg, Saikko, and Järvisalo, 2015a] [Berg, Saikko, and Järvisalo, 2016]
  - *How to simplify* MaxSAT *instances to make them easier for solver(s)?*
- Parallel MaxSAT solving [Martins, Manquinho, and Lynce, 2012] [Martins, Manquinho, and Lynce, 2015]
  - *How employ computing clusters to speed-up* MaxSAT *solving?*
- Variants and generalization
  - MinSAT [Li, Zhu, Manyà, and Simon, 2012] [Argelich, Li, Manyà, and Zhu, 2013] [Ignatiev, Morgado, Planes, and Marques-Silva, 2013b] [Li and Manyà, 2015]
  - Quantified MaxSAT [Ignatiev, Janota, and Marques-Silva, 2013a]

Further Topics

- instance decompositioning/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]
- ...

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 http://maxsat.ia.udl.cat

Overview articles: [Argelich, Li, Manyà, and Planes, 2008b]
[Argelich, Li, Manyà, and Planes, 2011]

Thank you for your attention!

Bibliography I

Mario Alviano, Carmine Dodaro, and Francesco Ricca. A maxsat algorithm using cardinality constraints of bounded size. In Yang and Wooldridge [2015], pages 2677–2683. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-379.html.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSat through satisfiability testing. In Kullmann [2009], pages 427–440. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_39. URL http://dx.doi.org/10.1007/978-3-642-02777-2_39.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSat algorithms. *Artif. Intell.*, 196:77–105, 2013a. doi: 10.1016/j.artint.2013.01.002. URL http://dx.doi.org/10.1016/j.artint.2013.01.002.

Carlos Ansótegui, Idelfonso Izquierdo, Felip Manyà, and José Torres-Jiménez. A max-sat-based approach to constructing optimal covering arrays. In Karina Gibert, Vicent J. Botti, and Ramón Reig Bolaño, editors, *Artificial Intelligence Research and Development - Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence, Vic, Catalonia, Spain, October 23-25, 2013.*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 51–59. IOS Press, 2013b. ISBN 978-1-61499-319-3. doi: 10.3233/978-1-61499-320-9-51. URL http://dx.doi.org/10.3233/978-1-61499-320-9-51.

Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in MaxSat. In Yang and Wooldridge [2015], pages 283–289. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-046.html.

Josep Argelich, Chu Min Li, and Felip Manyà. A preprocessor for max-sat solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 15–20. Springer, 2008a. ISBN 978-3-540-79718-0. doi: 10.1007/978-3-540-79719-7_2. URL http://dx.doi.org/10.1007/978-3-540-79719-7_2.

Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second max-sat evaluations. *JSAT*, 4(2-4):251–278, 2008b. URL http://jsat.ewi.tudelft.nl/content/volume4/JSAT4_13_Argelich.pdf.

Josep Argelich, Daniel Le Berre, Inês Lynce, João P. Marques-Silva, and Pascal Rapicault. Solving linux upgradeability problems using boolean optimization. In Inês Lynce and Ralf Treinen, editors, *Proceedings First International Workshop on Logics for Component Configuration, LoCoCo 2010, Edinburgh, UK, 10th July 2010.*, volume 29 of *EPTCS*, pages 11–22, 2010. doi: 10.4204/EPTCS.29.2. URL http://dx.doi.org/10.4204/EPTCS.29.2.

Bibliography II

Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Experimenting with the instances of the MaxSat evaluation. In Cèsar Fernández, Hector Geffner, and Felip Manyà, editors, *Artificial Intelligence Research and Development - Proceedings of the 14th International Conference of the Catalan Association for Artificial Intelligence, Lleida, Catalonia, Spain, October 26-28, 2011*, volume 232 of *Frontiers in Artificial Intelligence and Applications*, pages 31–40. IOS Press, 2011. ISBN 978-1-60750-841-0. doi: 10.3233/978-1-60750-842-7-31. URL http://dx.doi.org/10.3233/978-1-60750-842-7-31.

Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. Efficient encodings from CSP into sat, and from maxcsp into MaxSat. *Multiple-Valued Logic and Soft Computing*, 19(1-3):3–23, 2012. URL http://www.oldcitypublishing.com/MVLSC/MVLSCabstracts/MVLSC18.5-6abstracts/MVLSCv18n5-6p445-456Zou.html.

Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. Minsat versus MaxSat for optimization problems. In Schulte [2013], pages 133–142. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0_13. URL http://dx.doi.org/10.1007/978-3-642-40627-0_13.

Fahiem Bacchus and Nina Narodytska. Cores in core based MaxSat algorithms: An analysis. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 7–15. Springer, 2014. ISBN 978-3-319-09283-6. doi: 10.1007/978-3-319-09284-3_2. URL http://dx.doi.org/10.1007/978-3-319-09284-3_2.

Anton Belov, António Morgado, and João Marques-Silva. Sat-based preprocessing for MaxSat. In McMillan et al. [2013], pages 96–111. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5_7. URL http://dx.doi.org/10.1007/978-3-642-45221-5_7.

Jeremias Berg and Matti Järvisalo. Optimal correlation clustering via MaxSat. In Wei Ding, Takashi Washio, Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, pages 750–757. IEEE Computer Society, 2013. ISBN 978-0-7695-5109-8. doi: 10.1109/ICDMW.2013.99. URL http://dx.doi.org/10.1109/ICDMW.2013.99.

Jeremias Berg and Matti Järvisalo. Sat-based approaches to treewidth computation: An evaluation. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 328–335. IEEE Computer Society, 2014. ISBN 978-1-4799-6572-4. doi: 10.1109/ICTAI.2014.57. URL http://dx.doi.org/10.1109/ICTAI.2014.57.

Bibliography III

Jeremias Berg and Matti Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 2016a.

Jeremias Berg and Matti Järvisalo. Impact of SAT-based preprocessing on core-guided MaxSAT solving. In *Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP 2016)*, Lecture Notes in Computer Science. Springer, 2016b.

Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Proceedings*, pages 86–95. JMLR.org, 2014. URL `http://jmlr.org/proceedings/papers/v33/berg14.html`.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Re-using auxiliary variables for MaxSat preprocessing. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 813–820. IEEE, 2015a. ISBN 978-1-5090-0163-7. doi: 10.1109/ICTAI.2015.120. URL `http://dx.doi.org/10.1109/ICTAI.2015.120`.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Improving the effectiveness of sat-based preprocessing for MaxSat. In Yang and Wooldridge [2015], pages 239–245. ISBN 978-1-57735-738-4. URL `http://ijcai.org/papers15/Abstracts/IJCAI15-040.html`.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Subsumed label elimination for maximum satisfiability. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, Frontiers in Artificial Intelligence and Applications. IOS Press, 2016.

David Bergman, André A. Ciré, Willem Jan van Hoeve, and Tallys H. Yunes. Bdd-based heuristics for binary optimization. *J. Heuristics*, 20(2):211–234, 2014. doi: 10.1007/s10732-014-9238-1. URL `http://dx.doi.org/10.1007/s10732-014-9238-1`.

Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artif. Intell.*, 171(8-9):606–618, 2007. doi: 10.1016/j.artint.2007.03.001. URL `http://dx.doi.org/10.1016/j.artint.2007.03.001`.

Carla E. Brodley and Peter Stone, editors. *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*, 2014. AAAI Press. ISBN 978-1-57735-661-5. URL `http://www.aaai.org/Library/AAAI/aaai14contents.php`.

Bibliography IV

Kerstin Bunte, Matti Järvisalo, Jeremias Berg, Petri Myllymäki, Jaakko Peltonen, and Samuel Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In Brodley and Stone [2014], pages 1694–1700. ISBN 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8242.

Yibin Chen, Sean Safarpour, Andreas G. Veneris, and João P. Marques-Silva. Spatial and temporal design debug using partial MaxSat. In Fabrizio Lombardi, Sanjukta Bhanja, Yehia Massoud, and R. Iris Bahar, editors, *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*, pages 345–350. ACM, 2009. ISBN 978-1-60558-522-2. doi: 10.1145/1531542.1531621. URL http://doi.acm.org/10.1145/1531542.1531621.

Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 29(11):1804–1817, 2010. doi: 10.1109/TCAD.2010.2061270. URL http://dx.doi.org/10.1109/TCAD.2010.2061270.

Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. ISBN 978-3-642-23785-0. doi: 10.1007/978-3-642-23786-7_19. URL http://dx.doi.org/10.1007/978-3-642-23786-7_19.

Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Schulte [2013], pages 247–262. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0_21. URL http://dx.doi.org/10.1007/978-3-642-40627-0_21.

Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSat. In Järvisalo and Gelder [2013], pages 166–181. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_13. URL http://dx.doi.org/10.1007/978-3-642-39071-5_13.

Jessica Davies, Jeremy Cho, and Fahiem Bacchus. Using learnt clauses in MaxSat. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010. ISBN 978-3-642-15395-2. doi: 10.1007/978-3-642-15396-9_17. URL http://dx.doi.org/10.1007/978-3-642-15396-9_17.

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003. doi: 10.1145/636865.636866. URL http://doi.acm.org/10.1145/636865.636866.

Bibliography V

Zhiwen Fang, Chu-Min Li, Kan Qiao, Xu Feng, and Ke Xu. Solving maximum weight clique using maximum satisfiability reasoning. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *ECAI 2014 - 21st European Conference on Artificial Intelligence, 18-22 August 2014, Prague, Czech Republic - Including Prestigious Applications of Intelligent Systems (PAIS 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 303–308. IOS Press, 2014. ISBN 978-1-61499-418-3. doi: 10.3233/978-1-61499-419-0-303. URL http://dx.doi.org/10.3233/978-1-61499-419-0-303.

Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006. ISBN 3-540-37206-7. doi: 10.1007/11814948_25. URL http://dx.doi.org/10.1007/11814948_25.

João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In Milano [2012], pages 941–956. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7_67. URL http://dx.doi.org/10.1007/978-3-642-33558-7_67.

Federico Heras, António Morgado, and João Marques-Silva. Lower bounds and upper bounds for MaxSat. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, volume 7219 of *Lecture Notes in Computer Science*, pages 402–407. Springer, 2012. ISBN 978-3-642-34412-1. doi: 10.1007/978-3-642-34413-8_35. URL http://dx.doi.org/10.1007/978-3-642-34413-8_35.

Federico Heras, António Morgado, and Joao Marques-Silva. MaxSat-based encodings for group MaxSat. *AI Commun.*, 28(2): 195–214, 2015. doi: 10.3233/AIC-140636. URL http://dx.doi.org/10.3233/AIC-140636.

Antti Hyttinen, Frederick Eberhardt, and Matti Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In Nevin L. Zhang and Jin Tian, editors, *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 340–349. AUAI Press, 2014. ISBN 978-0-9749039-1-0. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2469&proceeding_id=30.

Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Quantified maximum satisfiability: - A core-guided approach. In Järvisalo and Gelder [2013], pages 250–266. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_19. URL http://dx.doi.org/10.1007/978-3-642-39071-5_19.

Bibliography VI

Alexey Ignatiev, António Morgado, Jordi Planes, and João Marques-Silva. Maximal falsifiability - definitions, algorithms, and applications. In McMillan et al. [2013], pages 439–456. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5_30. URL http://dx.doi.org/10.1007/978-3-642-45221-5_30.

Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Towards efficient optimization in package management systems. In Pankaj Jalote, Lionel C. Briand, and André van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 745–755. ACM, 2014. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568306. URL http://doi.acm.org/10.1145/2568225.2568306.

Matti Järvisalo and Allen Van Gelder, editors. *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, 2013. Springer. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5. URL http://dx.doi.org/10.1007/978-3-642-39071-5.

M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Proc. PLDI*, pages 437–446. ACM, 2011.

Oliver Kullmann, editor. *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, 2009. Springer. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2. URL http://dx.doi.org/10.1007/978-3-642-02777-2.

Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 193–198. Professional Book Center, 2005. ISBN 0938075934. URL http://dblp.uni-trier.de/db/conf/ijcai/ijcai2005.html#LarrosaH05.

Chu Min Li and Felip Manyà. MaxSat, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009. ISBN 978-1-58603-929-5. doi: 10.3233/978-1-58603-929-5-613. URL http://dx.doi.org/10.3233/978-1-58603-929-5-613.

Chu Min Li and Felip Manyà. An exact inference scheme for minsat. In Yang and Wooldridge [2015], pages 1959–1965. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-278.html.

Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1611.

Bibliography VII

Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 86–91. AAAI Press, 2006. URL http://www.aaai.org/Library/AAAI/2006/aaai06-014.php.

Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in max-sat. In Kullmann [2009], pages 467–480. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_43. URL http://dx.doi.org/10.1007/978-3-642-02777-2_43.

Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in maxsat. *Constraints*, 15(4):456–484, 2010. doi: 10.1007/s10601-010-9097-9. URL http://dx.doi.org/10.1007/s10601-010-9097-9.

Chu Min Li, Zhu Zhu, Felip Manyà, and Laurent Simon. Optimizing with minimum satisfiability. *Artif. Intell.*, 190:32–44, 2012. doi: 10.1016/j.artint.2012.05.004. URL http://dx.doi.org/10.1016/j.artint.2012.05.004.

Chu-Min Li, Hua Jiang, and Ruchu Xu. Incremental MaxSat reasoning to reduce branches in a branch-and-bound algorithm for maxclique. In Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion, editors, *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994 of *Lecture Notes in Computer Science*, pages 268–274. Springer, 2015. ISBN 978-3-319-19083-9. doi: 10.1007/978-3-319-19084-6_26. URL http://dx.doi.org/10.1007/978-3-319-19084-6_26.

Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in max-sat solving. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 351–356. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://www.aaai.org/Library/AAAI/2008/aaai08-055.php.

Inês Lynce and João Marques-Silva. Restoring CSP satisfiability with MaxSat. *Fundam. Inform.*, 107(2-3):249–266, 2011. doi: 10.3233/FI-2011-402. URL http://dx.doi.org/10.3233/FI-2011-402.

Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Kullmann [2009], pages 495–508. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_45. URL http://dx.doi.org/10.1007/978-3-642-02777-2_45.

João Marques-Silva and Jordi Planes. On using unsatisfiability for solving maximum satisfiability. *CoRR*, abs/0712.1097, 2007. URL http://arxiv.org/abs/0712.1097.

Bibliography VIII

João Marques-Silva, Inês Lynce, and Vasco M. Manquinho. Symmetry breaking for maximum satisfiability. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008. ISBN 978-3-540-89438-4. doi: 10.1007/978-3-540-89439-1_1. URL http://dx.doi.org/10.1007/978-3-540-89439-1_1.

João Marques-Silva, Mikolás Janota, Alexey Ignatiev, and António Morgado. Efficient model based diagnosis with maximum satisfiability. In Yang and Wooldridge [2015], pages 1966–1972. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-279.html.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Parallel search for maximum satisfiability. *AI Commun.*, 25(2):75–95, 2012. doi: 10.3233/AIC-2012-0517. URL http://dx.doi.org/10.3233/AIC-2012-0517.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Community-based partitioning for MaxSat solving. In Järvisalo and Gelder [2013], pages 182–191. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_14. URL http://dx.doi.org/10.1007/978-3-642-39071-5_14.

Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSat. In O'Sullivan [2014], pages 531–548. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_39. URL http://dx.doi.org/10.1007/978-3-319-10428-7_39.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Deterministic parallel MaxSat solving. *International Journal on Artificial Intelligence Tools*, 24(3), 2015. doi: 10.1142/s0218213015500050. URL http://dx.doi.org/10.1142/s0218213015500050.

Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors. *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, 2013. Springer. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5. URL http://dx.doi.org/10.1007/978-3-642-45221-5.

Michela Milano, editor. *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, 2012. Springer. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7. URL http://dx.doi.org/10.1007/978-3-642-33558-7.

Bibliography IX

Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013. doi: 10.1287/opre.1120.1139. URL http://dx.doi.org/10.1287/opre.1120.1139.

António Morgado and João Marques-Silva. Combinatorial optimization solutions for the maximum quartet consistency problem. *Fundam. Inform.*, 102(3-4):363–389, 2010. doi: 10.3233/FI-2010-311. URL http://dx.doi.org/10.3233/FI-2010-311.

António Morgado, Mark H. Liffiton, and João Marques-Silva. MaxSat-based MCS enumeration. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2012. ISBN 978-3-642-39610-6. doi: 10.1007/978-3-642-39611-3_13. URL http://dx.doi.org/10.1007/978-3-642-39611-3_13.

António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. Iterative and core-guided MaxSat solving: A survey and assessment. *Constraints*, 18(4):478–534, 2013. doi: 10.1007/s10601-013-9146-2. URL http://dx.doi.org/10.1007/s10601-013-9146-2.

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.

Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSat resolution. In Brodley and Stone [2014], pages 2717–2723. ISBN 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513.

Miguel Neves, Ruben Martins, Mikolás Janota, Inês Lynce, and Vasco M. Manquinho. Exploiting resolution-based representations for MaxSat solving. In Marijn Heule and Sean Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 272–286. Springer, 2015. ISBN 978-3-319-24317-7. doi: 10.1007/978-3-319-24318-4_20. URL http://dx.doi.org/10.1007/978-3-319-24318-4_20.

Barry O'Sullivan, editor. *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, 2014. Springer. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7. URL http://dx.doi.org/10.1007/978-3-319-10428-7.

Bibliography X

James D. Park. Using weighted MAX-SAT engines to solve MPE. In Rina Dechter and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 682–687. AAAI Press / The MIT Press, 2002. URL http://www.aaai.org/Library/AAAI/2002/aaai02-102.php.

Paul Saikko, Brandon Malone, and Matti Järvisalo. MaxSat-based cutting planes for learning graphical models. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 347–356. Springer, 2015. ISBN 978-3-319-18007-6. doi: 10.1007/978-3-319-18008-3_24. URL http://dx.doi.org/10.1007/978-3-319-18008-3_24.

Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In Nadia Creignou and Daniel Le Berre, editors, *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT 2016)*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

Christian Schulte, editor. *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, 2013. Springer. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0. URL http://dx.doi.org/10.1007/978-3-642-40627-0.

Johannes Peter Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages ???–??? AAAI Press, 2016.

Qiang Yang and Michael Wooldridge, editors. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, 2015. AAAI Press. ISBN 978-1-57735-738-4.

Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada.* AAAI Press, 2012. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5190.

Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 63–66. FMCAD Inc., 2011. ISBN 978-0-9835678-1-3. URL http://dl.acm.org/citation.cfm?id=2157667.

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.