MaxSAT Latest Developments
CP13

Carlos Ansótegui

Área: Ciencias de la Computación e Inteligencia Artificial (CCIA)
Departamento de Informática e Ingeniería Industrial
Escuela Universitaria Politécnica
Universidad de Lleida

September 2013

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Some NP-Complete Problems:
Gary and Johnson, 1979

Vertex cover, Dominating set, Domatic number, Graph k-colorability, Achromatic number, Monochromatic triangle, Feedback vertex set, Feedback arc set, Partial feedback edge set, Minimum maximal matching, Partition into triangles, Partition into isomorphic subgraphs, Partition into Hamiltonian subgraphs, Partition into forests, Partition into cliques, Partition into perfect matchings, Two-stage maximum weight stochastic matching, Covering by cliques, Covering by complete bipartite subgraphs, Clique, Independent set, Induced subgraph with property π, Induced connected subgraph with property π, Induced path, Balanced complete bipartite subgraph, Bipartite subgraph, Degree-bounded connected subgraph, Planar subgraph, Edge-subgraph, Transitive subgraph, Uniconnected subgraph, Minimum k-connected subgraph, Cubic subgraph, Minimum equivalent digraph, Hamiltonian completion, Interval graph completion,

Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Path graph completion, Hamiltonian circuit, Directed Hamiltonian circuit, Hamiltonian path, Bandwidth, Directed bandwidth, Optimal linear arrangement, Directed optimal linear arrangement, Minimum cut linear arrangement, Rooted tree arrangement, Directed elimination ordering, Elimination degree sequence, Subgraph isomorphism, Largest common subgraph, Maximum subgraph matching, Graph contractability, Graph homomorphism, Digraph D-morphism, Path with forbidden pairs, Multiple choice matching, Graph Grundy numbering, Kernel, K-closure, Intersection graph basis, Path distinguishers, Metric dimension, Nesetril-Rodl dimension, Threshold number, Oriented diameter, Weighted diameter, Degree constrained spanning tree, Maximum leaf spanning tree, Shortest total path length spanning tree, Bounded diameter spanning tree, Capacitated spanning tree,

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

   Geometric capacitated spanning tree, Optimum communication spanning tree, Isomorphic spanning tree, Kth best spanning tree, Bounded component spanning forest, Multiple choice branching, Steiner tree, Geometric Steiner tree, Cable Trench Problem, Graph partitioning, Acyclic partition, Max weight cut, Minimum cut into bounded sets, Biconnectivity augmentation, Strong connectivity augmentation, Network reliability, Network survivability, Multiway Cut, Minimum k-cut, Bottleneck traveling salesman, Chinese postman for mixed graphs, Euclidean traveling salesman, K most vital arcs, Kth shortest path, Metric traveling salesman, Longest circuit, Longest path, Prize Collecting Traveling Salesman, Rural Postman, Shortest path in general networks, Shortest weight-constrained path, Stacker-crane, Time constrained traveling salesman feasibility, Traveling salesman problem, Vehicle Routing, Minimum edge-cost flow,

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Integral flow with multipliers, Path constrained network flow, Integral flow with homologous arcs, Integral flow with bundles, Undirected flow with lower bounds, Directed two-commodity integral flow, Undirected two-commodity integral flow, Disjoint connecting paths, Maximum length-bounded disjoint paths, Maximum fixed-length disjoint paths, Unsplittable multicommodity flow Quadratic assignment problem, Minimizing dummy activities in PERT networks, Constrained triangulation, Intersection graph for segments on a grid, Edge embedding on a grid, Geometric connected dominating set, Minimum broadcast time, Min-max multicenter, Min-sum multicenter, Uncapacitated Facility Location, Metric k-center, 3-dimensional matching, Exact cover, Set packing, Set splitting, Set cover, Minimum test set, Set basis, Hitting set, Intersection pattern, Comparative containment, 3-matroid intersection, Partition, Subset sum, Subset product, 3-partition, Numerical 3-dimensional matching, Numerical matching with target sums, Expected component sum, Minimum sum of squares, Kth largest subset, Kth largest m-tuple, Bin packing, Dynamic storage allocation,

Why Satisfiability (SAT)?

Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Pruned trie space minimization, Expected retrieval cost, Rooted tree storage assignment, Multiple copy file allocation, Capacity assignment, Shortest common supersequence, Shortest common superstring, Longest common subsequence problem for the case of arbitrary (i.e., not a priori fixed) number of input sequences even in the case of the binary alphabet, Bounded post correspondence problem, Hitting string, Sparse matrix compression, Consecutive ones submatrix, Consecutive ones matrix partition, Consecutive ones matrix augmentation, Consecutive block minimization, Consecutive sets, 2-dimensional consecutive sets, String-to-string correction, Grouping by swapping, External macro data compression, Internal macro data compression, Regular expression substitution, Rectilinear picture compression, Optimal vector quantization codebook, Minimal grammar-based compression, Minimum cardinality key, Additional key, Prime attribute name, Boyce-Codd normal form violation, Conjunctive query foldability, Conjunctive boolean query, Tableau equivalence,

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Serializability of database histories, Safety of database transaction systems, Consistency of database frequency tables, Safety of file protection systems, Sequencing with release times and deadlines, Sequencing to minimize Tardy tasks, Sequencing to minimize Tardy weight, Sequencing to minimize weighted completion time, Sequencing to minimize weighted tardiness, Sequencing with deadlines and set-up times, Sequencing to minimize maximum cumulative cost, Multiprocessor scheduling, Precedence constrained scheduling, Resource constrained scheduling, Scheduling with individual deadlines, Preemptive scheduling, Scheduling to minimize weighted completion time, Open-shop scheduling, Flow-shop scheduling, No-wait flow-shop scheduling, Two-processor flow-shop with bounded buffer, Job-shop scheduling, Timetable design, Staff scheduling, Production planning, Deadlock avoidance,

Why Satisfiability (SAT)?

▸ Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding

Integer programming, 0-1 Integer programming, Quadratic programming (NP-hard in some cases, P when convex), Cost-parametric linear programming, Feasible basis extension, Minimum weight solution to linear equations, Open hemisphere, K-relevancy, Traveling salesman polytope non-adjacency, Knapsack, Integer knapsack, Continuous multiple choice knapsack, Partially ordered knapsack, Comparative vector inequalities

Why Satisfiability (SAT)?

Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding
3. Specialized algorithms can be difficult to develop. Alternative: translate the problem into SAT and run a SAT solver

Why Satisfiability (SAT)?

Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding
3. Specialized algorithms can be difficult to develop. Alternative: translate the problem into SAT and run a SAT solver
4. Bad news: not known polytime algorithm for SAT

   Good news: however, worst-case often does not show up!

Why Satisfiability (SAT)?

Why MaxSAT

1. SAT is an NP-Complete decision problem. SAT was the first problem to be shown NP-Complete [Cook, 1971]
2. All NP problems can be (polynomially) reduced to SAT. Key issue: find an efficient encoding
3. Specialized algorithms can be difficult to develop. Alternative: translate the problem into SAT and run a SAT solver
4. Bad news: not known polytime algorithm for SAT

   Good news: however, worst-case often does not show up!
5. Current SAT solvers solve instances with millions of clauses and hundreds of thousands of Boolean variables in seconds
6. In the last decade SAT has seen many great advances, including efficient inference, non-chronological backtracking, conflict driven clause learning and unsat core generation

- Formal methods: Hardware model checking; Software model checking; Termination analysis of term-rewrite systems; Test pattern generation (testing of software & hardware); etc.
- Artificial intelligence: Planning; Knowledge representation; Games (n-queens, sudoku, social golpher, etc.)
- Bioinformatics: Haplotype inference; Pedigree checking; Comparative genomics; etc.
- Design automation: Equivalence checking; Delay computation; Fault diagnosis; Noise analysis; etc.
- Security: Cryptanalysis; Inversion attacks on hash functions; etc.
- Computationally hard problems: Graph coloring; Traveling salesperson; etc.
- Mathematical problems: van der Waerden numbers; etc.
- Core engine for other solvers: PB; QBF; #SAT; SMT; MaxSAT ...
- Integrated into theorem provers: HOL; Isabelle; ...

Solving hard combinatorial decision problems

Combinatorial decision problem

$\downarrow$

Reduction (translation) to SAT

$\downarrow$

SAT solver

$\downarrow$

Solution

The SAT problem consists in determining whether a Boolean formula in conjunctive normal form is satisfiable
The answer is yes/no (satisfiable/unsatisfiable).

Syntax of a SAT formula

Given a set of Boolean variables $Var = \{x_1, x_2, \ldots, x_n\}$, we define:

- literal l:
  as a variable in positive form x_i or negative $\overline{x}_i$
- clause C:
  as a disjunction of literals, $l_1 \vee \ldots \vee l_k$
- SAT formula:
  as a conjunction of clauses, $C_1 \wedge \ldots \wedge C_n$

Syntax of a SAT formula

Given a set of Boolean variables $Var = \{x_1, x_2, \ldots, x_n\}$, we define:

- literal l:
  as a variable in positive form x_i or negative $\overline{x}_i$
- clause C:
  as a disjunction of literals, $l_1 \vee \ldots \vee l_k$
- SAT formula:
  as a conjunction of clauses, $C_1 \wedge \ldots \wedge C_n$

Example SAT formula

Translate ($x_1 = 1$ and $x_1 + x_2 = 1$) into SAT, where $x_i \in \{0, 1\}$

$$(x_1) \wedge (x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$$

Syntax of a SAT formula

Given a set of Boolean variables $Var = \{x_1, x_2, \ldots, x_n\}$, we define:

- literal l:
  as a variable in positive form x_i or negative $\overline{x}_i$
- clause C:
  as a disjunction of literals, $l_1 \vee \ldots \vee l_k$
- SAT formula:
  as a conjunction of clauses, $C_1 \wedge \ldots \wedge C_n$

Special symbols

$\Box$ denotes the *empty clause*
$\emptyset$ denotes the *empty formula*

Semantics of a SAT formula

An interpretation $\mathcal{I}$ is a function from Var to $\{0, 1\}^n$.

$\mathcal{I}$ satisfies:

- a literal x_i iff $\mathcal{I}(x_i) = 1$
- a literal $\overline{x}_i$ iff $\mathcal{I}(x_i) = 0$
- a clause C iff $\mathcal{I}$ satisfies at least one of its literals
- a SAT formula iff $\mathcal{I}$ satisfies all its clauses

A SAT formula is satisfiable iff there is an interpretation which satisfies the formula. Otherwise, it is unsatisfiable.

Semantics of a SAT formula

An interpretation $\mathcal{I}$ is a function from Var to $\{0,1\}^n$.

$\mathcal{I}$ satisfies:

- a literal x_i iff $\mathcal{I}(x_i) = 1$
- a literal $\overline{x}_i$ iff $\mathcal{I}(x_i) = 0$
- a clause C iff $\mathcal{I}$ satisfies at least one of its literals
- a SAT formula iff $\mathcal{I}$ satisfies all its clauses

Observation:

Be φ a SAT formula, then

If $\square \in \varphi$, φ is trivially unsatisfiable

If $\varphi = \emptyset$, φ is trivially satisfiable

Semantics of a SAT formula

An interpretation $\mathcal{I}$ is a function from Var to $\{0,1\}^n$.

$\mathcal{I}$ satisfies:

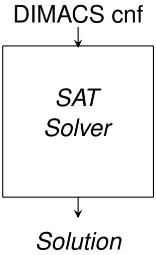
- a literal x_i iff $\mathcal{I}(x_i) = 1$
- a literal $\overline{x}_i$ iff $\mathcal{I}(x_i) = 0$
- a clause C iff $\mathcal{I}$ satisfies at least one of its literals
- a SAT formula iff $\mathcal{I}$ satisfies all its clauses

Example truth table

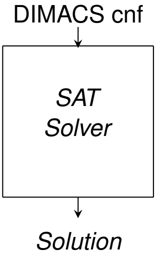
$$\varphi = (x_1) \wedge (x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$$

$\mathcal{I}(x_1)$	$\mathcal{I}(x_2)$	$\mathcal{I}(\varphi)$
0	0	0
0	1	0
1	0	1
1	1	0

Using SAT solvers



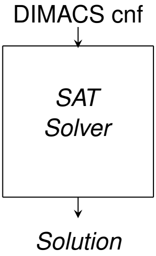
Using SAT solvers



Example formulation

$$(x_1)$$
$$(x_1 \vee x_2)$$
$$(\overline{x}_1 \vee \overline{x}_2)$$

Using SAT solvers



Example formulation

$$(x_1)$$
$$(x_1 \vee x_2)$$
$$(\overline{x}_1 \vee \overline{x}_2)$$

Formula in DIMACS cnf

```
p cnf 2 3
 1   0
 1   2   0
-1  -2   0
```

Using SAT solvers

DIMACS cnf

↓

SAT Solver

↓

Solution

Execution output

s SATISFIABLE
v 1 -2 0

Example formulation

$$(x_1)$$
$$(x_1 \vee x_2)$$
$$(\overline{x}_1 \vee \overline{x}_2)$$

Formula in DIMACS cnf

```
p cnf 2 3
 1    0
 1    2    0
-1   -2    0
```

Why Maximum Satisfiability (MaxSAT)?

- SAT solvers can solve efficiently several industrial/real problems
- MaxSAT is the optimization version of SAT
- MaxSAT can be an efficient approach for industrial/real optimization problems
- MaxSAT technology can be extended to richer formalisms, such as: Weighted CSP or Weighted SMT
- Great opportunity for hybrid SAT and OR approaches

Practical Applications of MaxSAT

- Bioinformatics: haplotyping-pedigrees
- Software: package installation, covering arrays
- Telecommunications: frequency assignment
- Scheduling: satellite scheduling, timetabling, rcpsp
- Circuits design: debugging, logic synthesis
- Electronic markets: combinatorial auctions

Solving hard combinatorial optimization problems

Combinatorial optimization problem

$\downarrow$

Reduction (translation) to MaxSAT

$\downarrow$

MaxSAT solver

$\downarrow$

Solution

The MaxSAT problem consist in determining the maximum number of clauses that can be satisfied

Combinatorial optimization problem

↓

Reduction (translation) to MaxSAT

↓

MaxSAT solver

↓

Solution

The MaxSAT can be solved by determining the minimum number of clauses that can be unsatisfied (violated/falsified)

SAT vs. MaxSAT

Decisional MaxSAT

input: multiset of clauses and an integer k
output: yes/no whether there is an assignment that falsifies $\leq k$ clauses

NP-complete

Optimization MaxSAT

input: multiset of clauses
output: an assignment that falsifies the minimum number of clauses

MAX SNP-complete

MaxSAT formula

$\varphi = (x_1) \wedge (x_2) \wedge (x_1 \vee x_2) \wedge (\overline{x}_1 \vee \overline{x}_2)$

$\mathcal{I}(x_1)$	$\mathcal{I}(x_2)$	$\mathcal{I}(\varphi)$
0	0	3
0	1	1
1	0	1
1	1	1

$\mathcal{I}(\varphi) =$ num of unsatisfied clauses in φ under $\mathcal{I}$

Weighted MaxSAT

Weighted MaxSAT formula

$$\varphi = (x_1, 1) \wedge (x_2, 1) \wedge (x_1 \vee x_2, 1) \wedge (\overline{x}_1 \vee \overline{x}_2, 1)$$

$\mathcal{I}(x_1)$	$\mathcal{I}(x_2)$	$\mathcal{I}(\varphi)$
0	0	3
0	1	1
1	0	1
1	1	1

$\mathcal{I}(\varphi) =$ num of unsatisfied clauses in φ under $\mathcal{I}$

A *Weighted* clause is a pair (C, w), where C is a clause and w is a natural number or infinite (that indicates the cost of falsifying C).

We refer to the formulas that contain *Weighted* clauses as *Weighted*.

Weighted MaxSAT

Weighted MaxSAT formula

$$\varphi = (x_1, 2) \wedge (x_2, 1) \wedge (x_1 \vee x_2, 1) \wedge (\overline{x}_1 \vee \overline{x}_2, 1)$$

$\mathcal{I}(x_1)$	$\mathcal{I}(x_2)$	$\mathcal{I}(\varphi)$
0	0	4
0	1	2
1	0	1
1	1	1

$\mathcal{I}(\varphi) =$ num of unsatisfied clauses in φ under $\mathcal{I}$

A *Weighted* clause is a pair (C, w), where C is a clause and w is a natural number or infinite (that indicates the cost of falsifying C).

We refer to the formulas that contain *Weighted* clauses as *Weighted*.

Weighted Partial MaxSAT

Weighted Partial MaxSAT

$$\varphi = (x_1, 2) \wedge (x_2, 1) \wedge (x_1 \vee x_2, \infty) \wedge (\overline{x}_1 \vee \overline{x}_2, \infty)$$

$\mathcal{I}(x_1)$	$\mathcal{I}(x_2)$	$\mathcal{I}(\varphi)$
0	0	∞
0	1	2
1	0	1
1	1	∞

$\mathcal{I}(\varphi) =$ sum of weights of unsatisfied clauses in φ under $\mathcal{I}$

A clause is *hard* if the associated weight is infinite, otherwise the clause is *soft*.

We refer to the formulas that contain both types of clauses as *Partial*.

Weighted Partial MaxSAT: optimum cost

Given a Weighted Partial MaxSAT formula φ and an interpretation $\mathcal{I}$, the cost of $\mathcal{I}$ over φ, $\mathcal{I}(\varphi)$, is the sum of the weights of the clauses falsified by $\mathcal{I}$, i.e.

$$\mathcal{I}(\varphi) = \sum_{\substack{(C_i,w_i)\in\varphi \\ \mathcal{I}(C_i)=0}} w_i$$

The *optimum cost* of a formula is the minimum cost of all of its interpretations.

An *optimal interpretation* is an interpretation with optimum cost.

The Maximum Satisfiability problem of a MaxSAT formula consist in determining its optimum cost.

Weighted Partial MaxSAT: optimum cost

Given a Weighted Partial MaxSAT formula φ and an interpretation $\mathcal{I}$, the cost of $\mathcal{I}$ over φ, $\mathcal{I}(\varphi)$, is the sum of the weights of the clauses falsified by $\mathcal{I}$, i.e.

$$\mathcal{I}(\varphi) = \sum_{\substack{(C_i, w_i) \in \varphi \\ \mathcal{I}(C_i) = 0}} w_i$$

The *optimum cost* of a formula is the minimum cost of all of its interpretations.

An *optimal interpretation* is an interpretation with optimum cost.

The Maximum Satisfiability problem of a MaxSAT formula consist in determining its optimum cost.

Weighted Partial MaxSAT: optimum cost

Given a Weighted Partial MaxSAT formula φ and an interpretation $\mathcal{I}$, the cost of $\mathcal{I}$ over φ, $\mathcal{I}(\varphi)$, is the sum of the weights of the clauses falsified by $\mathcal{I}$, i.e.

$$\mathcal{I}(\varphi) = \sum_{\substack{(C_i, w_i) \in \varphi \\ \mathcal{I}(C_i) = 0}} w_i$$

The *optimum cost* of a formula is the minimum cost of all of its interpretations.

An *optimal interpretation* is an interpretation with optimum cost.

The Maximum Satisfiability problem of a MaxSAT formula consist in determining its optimum cost.

Weighted Partial MaxSAT: optimum cost

Given a Weighted Partial MaxSAT formula φ and an interpretation $\mathcal{I}$, the cost of $\mathcal{I}$ over φ, $\mathcal{I}(\varphi)$, is the sum of the weights of the clauses falsified by $\mathcal{I}$, i.e.

$$\mathcal{I}(\varphi) = \sum_{\substack{(C_i, w_i) \in \varphi \\ \mathcal{I}(C_i)=0}} w_i$$

The *optimum cost* of a formula is the minimum cost of all of its interpretations.

An *optimal interpretation* is an interpretation with optimum cost.

The Maximum Satisfiability problem of a MaxSAT formula consist in determining its optimum cost.

MaxSAT problems

Weighted Partial MaxSAT problem (WPMS)

The *Weighted Partial MaxSAT problem* for a weighted partial MaxSAT formula φ is the problem of finding an *optimal assignment*. If the optimal cost is infinity, we say that the formula is *unsatisfiable*.

Weighted MaxSAT problem (WMS)

The *Weighted MaxSAT problem* is the Weighted Partial MaxSAT problem when there are no hard clauses.

Partial MaxSAT problem (PMS)

The *Partial MaxSAT problem* is the Weighted Partial MaxSAT problem when the weights of soft clauses are equal.

MaxSAT problems

MaxSAT problem

The *MaxSAT problem* is the Partial MaxSAT problem when there are no hard clauses.

SAT problem

The *SAT problem* is equivalent to the Partial MaxSAT problem when there are no soft clauses.

WPMS as Integer Linear Programming (ILP)

The MaxSAT problem on a WPMS formula,

$$\{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$$

can be formulated as the following ILP problem,

$$\textbf{minimize: } b_1 \cdot w_1 + \cdots + b_m \cdot w_m \tag{1}$$

subject to the following constraints:

$$\bigwedge_{i=1}^{m} \overline{C_i} \rightarrow b_i \tag{2}$$

$$\bigwedge_{j=m+1}^{m'} C_j \tag{3}$$

Note: constraints are translated into linear inequalities in the regular way.

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

Maximum Clique

Instance: undirected graph $G = (V, E)$
Question: maximum set of vertices that is a clique in G?

$$\begin{array}{ll} \text{maximize} & |V'| \\ \text{subject to} & V' \subseteq V \\ & \forall_{v'_1, v'_2 \in V'} \{v'_1, v'_2\} \in E \end{array}$$

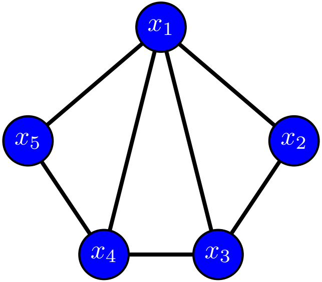
A *clique* in an undirected Graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices, each pair of which is connected by an edge in E

Maximum Clique as Partial MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

if $x_v = 1$, node v is in the clique



Standard formulation

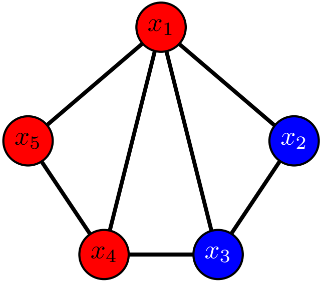
- Soft: for each node $v \in V$ we add, $(x_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \notin E$ we add, $(\overline{x}_{v_1} \vee \overline{x}_{v_2}, \infty)$

Maximum Clique as Partial MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

if $x_v = 1$, node v is in the clique



Standard formulation

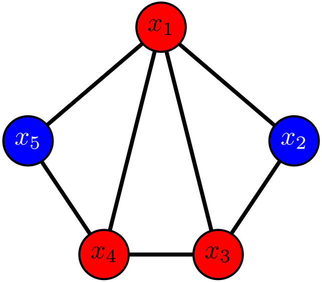
- Soft: for each node $v \in V$ we add, $(x_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \notin E$ we add, $(\overline{x}_{v_1} \vee \overline{x}_{v_2}, \infty)$

Maximum Clique as Partial MaxSAT

Boolean variables:

$X = \{x_v \mid v \in V\}$,

if $x_v = 1$, node v is in the clique



Standard formulation

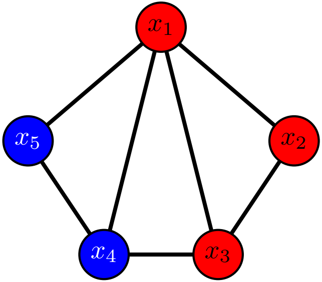
- Soft: for each node $v \in V$ we add, $(x_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \notin E$ we add, $(\overline{x}_{v_1} \vee \overline{x}_{v_2}, \infty)$

Maximum Clique as Partial MaxSAT

Boolean variables:

$X = \{x_v \mid v \in V\}$,

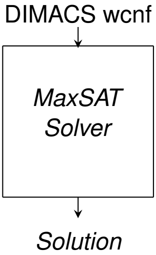
if $x_v = 1$, node v is in the clique



Standard formulation

- Soft: for each node $v \in V$ we add, $(x_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \notin E$ we add, $(\overline{x}_{v_1} \vee \overline{x}_{v_2}, \infty)$

Using MaxSAT solvers



Using MaxSAT solvers

DIMACS wcnf

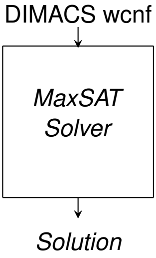
MaxSAT Solver

Solution

MaxClique formulation

$$(x_1, 1)$$
$$(x_2, 1)$$
$$(x_3, 1)$$
$$(x_4, 1)$$
$$(x_5, 1)$$
$$(\overline{x}_2 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_2 \vee \overline{x}_5, \infty)$$
$$(\overline{x}_3 \vee \overline{x}_5, \infty)$$

Using MaxSAT solvers

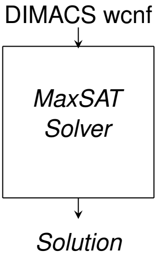


MaxClique formulation

$$(x_1, 1)$$
$$(x_2, 1)$$
$$(x_3, 1)$$
$$(x_4, 1)$$
$$(x_5, 1)$$
$$(\overline{x}_2 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_2 \vee \overline{x}_5, \infty)$$
$$(\overline{x}_3 \vee \overline{x}_5, \infty)$$

Formula in DIMACS wcnf

```
p wcnf 5 8 6
1   1   0
1   2   0
1   3   0
1   4   0
1   5   0
6  -2  -4   0
6  -2  -5   0
6  -3  -5   0
```



Execution output

```
o 2
s OPTIMUM FOUND
v 1 -2 -3 4 5 0
```

MaxClique formulation

$$
\begin{array}{l}
(x_1, 1) \\
(x_2, 1) \\
(x_3, 1) \\
(x_4, 1) \\
(x_5, 1) \\
(\overline{x}_2 \vee \overline{x}_4, \infty) \\
(\overline{x}_2 \vee \overline{x}_5, \infty) \\
(\overline{x}_3 \vee \overline{x}_5, \infty)
\end{array}
$$

Formula in DIMACS wcnf

```
p wcnf 5 8 6
1  1  0
1  2  0
1  3  0
1  4  0
1  5  0
6  -2  -4  0
6  -2  -5  0
6  -3  -5  0
```

Minimum Vertex Cover

Instance: undirected graph $G = (V, E)$
Question: minimum set of vertices that cover all edges?

$$\begin{array}{ll} \text{minimize} & |V'| \\ \text{subject to} & V' \subseteq V \\ & \forall \{v_1, v_2\} \in E : \{v_1, v_2\} \cap V' \neq \emptyset \end{array}$$

A *vertex cover* of an undirected graph $G = (V, E)$ is a subset $V' \subseteq V$ such that if $\{v_1, v_2\} \in E$, then $v_1 \in V'$ or $v_2 \in V'$ (or both).

Observation:

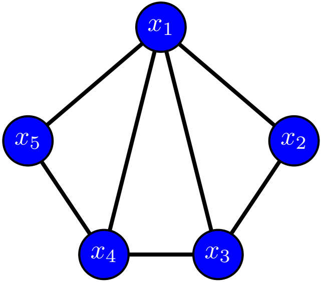
Maximum Clique(G) = Minimum Vertex Cover($\overline{G}$), where $\overline{G}$ is the complementary of G

Minimum Vertex Cover as Partial MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

if $x_v = 1$, node v is in the cover



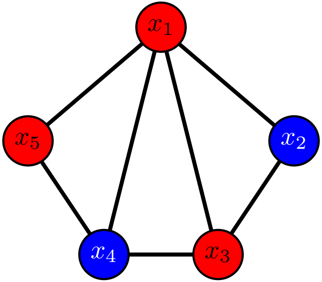
Standard formulation

- Soft: for each node $v \in V$ we add, $(\overline{x}_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \in E$ we add, $(x_{v_1} \vee x_{v_2}, \infty)$

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

if $x_v = 1$, node v is in the cover



Standard formulation

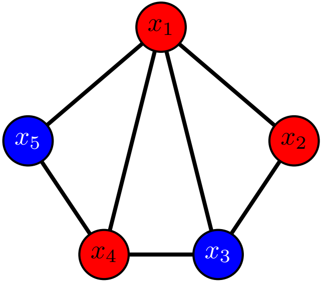
- Soft: for each node $v \in V$ we add, $(\overline{x}_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \in E$ we add, $(x_{v_1} \vee x_{v_2}, \infty)$

Minimum Vertex Cover as Partial MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

if $x_v = 1$, node v is in the cover



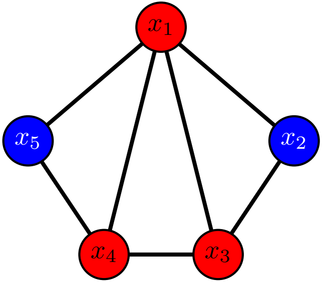
Standard formulation

- Soft: for each node $v \in V$ we add, $(\overline{x}_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \in E$ we add, $(x_{v_1} \vee x_{v_2}, \infty)$

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

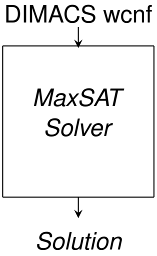
if $x_v = 1$, node v is in the cover



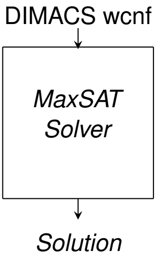
Standard formulation

- Soft: for each node $v \in V$ we add, $(\overline{x}_v, 1)$
- Hard: for each edge $\{v_1, v_2\} \in E$ we add, $(x_{v_1} \vee x_{v_2}, \infty)$

Using MaxSAT solvers



Using MaxSAT solvers



MinVertexCover formulation

$$
\begin{array}{l}
(\overline{x}_1, 1) \\
(\overline{x}_2, 1) \\
(\overline{x}_3, 1) \\
(\overline{x}_4, 1) \\
(\overline{x}_5, 1) \\
(x_1 \vee x_2, \infty) \\
(x_1 \vee x_3, \infty) \\
(x_2 \vee x_3, \infty) \\
(x_3 \vee x_4, \infty) \\
(x_4 \vee x_5, \infty) \\
(x_1 \vee x_4, \infty) \\
(x_1 \vee x_5, \infty)
\end{array}
$$

Using MaxSAT solvers

DIMACS wcnf

$\downarrow$

MaxSAT Solver

$\downarrow$

Solution

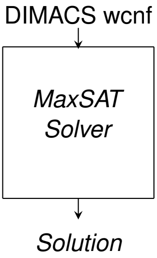
MinVertexCover formulation

$$
\begin{array}{l}
(\overline{x}_1, 1) \\
(\overline{x}_2, 1) \\
(\overline{x}_3, 1) \\
(\overline{x}_4, 1) \\
(\overline{x}_5, 1) \\
(x_1 \vee x_2, \infty) \\
(x_1 \vee x_3, \infty) \\
(x_2 \vee x_3, \infty) \\
(x_3 \vee x_4, \infty) \\
(x_4 \vee x_5, \infty) \\
(x_1 \vee x_4, \infty) \\
(x_1 \vee x_5, \infty)
\end{array}
$$

Formula in DIMACS wcnf

```
p wcnf 5 12 6
1   -1   0
1   -2   0
1   -3   0
1   -4   0
1   -5   0
6    1   2   0
6    1   3   0
6    2   3   0
6    3   4   0
6    4   5   0
6    1   4   0
6    1   5   0
```

Using MaxSAT solvers



Execution output

o 3
s OPTIMUM FOUND
v 1 -2 3 4 -5 0

MinVertexCover formulation

$$
\begin{array}{l}
(\overline{x}_1, 1) \\
(\overline{x}_2, 1) \\
(\overline{x}_3, 1) \\
(\overline{x}_4, 1) \\
(\overline{x}_5, 1) \\
(x_1 \vee x_2, \infty) \\
(x_1 \vee x_3, \infty) \\
(x_2 \vee x_3, \infty) \\
(x_3 \vee x_4, \infty) \\
(x_4 \vee x_5, \infty) \\
(x_1 \vee x_4, \infty) \\
(x_1 \vee x_5, \infty)
\end{array}
$$

Formula in DIMACS wcnf

```
p wcnf 5 12 6
1   -1   0
1   -2   0
1   -3   0
1   -4   0
1   -5   0
6    1   2   0
6    1   3   0
6    2   3   0
6    3   4   0
6    4   5   0
6    1   4   0
6    1   5   0
```

Maximum Cut problem

Instance: undirected graph $G = (V, E)$
Question: maximum cut in G?

$$\begin{array}{ll} \text{maximize} & |\{\{v_1, v_2\} \in E \mid v_1 \in S \wedge v_2 \in T\}| \\ \text{subject to} & S, T \subseteq V \\ & S \cup T = V \\ & S \cap T = \emptyset \end{array}$$

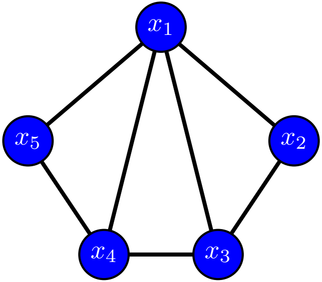
Given an undirected graph $G = (V, E)$, a *cut* (S, T) is a partition of V into two parts S and T.

Maximum Cut as MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

$x_v = 1$ if node $v \in S$
Otherwise, node
$v \in T$.



Standard formulation

- Soft: for each edge $\{v_1, v_2\} \in V$ we add,

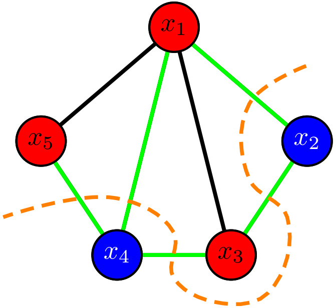
$$(x_{v_1} \vee x_{v_2}, 1) \wedge (\overline{x}_{v_1} \vee \overline{x}_{v_2}, 1)$$

Maximum Cut as MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

$x_v = 1$ if node $v \in S$
Otherwise, node
$v \in T$.



Standard formulation

- Soft: for each edge $\{v_1, v_2\} \in V$ we add,

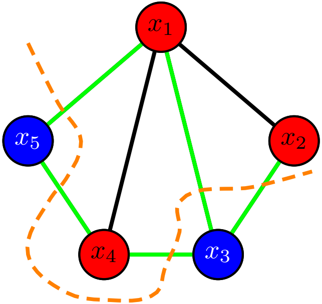
$$(x_{v_1} \vee x_{v_2}, 1) \wedge (\overline{x}_{v_1} \vee \overline{x}_{v_2}, 1)$$

Maximum Cut as MaxSAT

Boolean variables:

$$X = \{x_v \mid v \in V\},$$

$x_v = 1$ if node $v \in S$
Otherwise, node
$v \in T$.



Standard formulation

- Soft: for each edge $\{v_1, v_2\} \in V$ we add,

$$(x_{v_1} \vee x_{v_2}, 1) \wedge (\overline{x}_{v_1} \vee \overline{x}_{v_2}, 1)$$

Translating combinatorial auctions into MaxSAT

Instance: a combinatorial auction with,

- a set K of agents,
- a set N of goods, and
- a set M of bids of the form (S, w), where $S \subseteq N$ and $w \in \mathbb{N}$

Question: maximum benefit for the auctioneer?

Combinatorial Auctions as Weighted Partial MaxSAT

We define the following Boolean variables:

- $X = \{x_1, \ldots, x_M\}$,
  where $x_i = 1$ means the bid i is a winning bid.

Standard formulation [Larrosa et al., 2008]

- Soft (*cost of the bids*):
  For each bid i, we add,

$$(x_i, w_i)$$

- Hard (*incompatibility of bids*):
  For each pair of bids i and j, such that $S_i \cap S_j \neq \emptyset$, we add,

$$(\overline{x}_i \vee \overline{x}_j, \infty)$$

Combinatorial Auctions as Weighted Partial MaxSAT

Extended formulation

- Hard (*Minimum winning bids*):

  For each set of bids of an agent, $\{x_1, \ldots, x_k\}$, we add,

$$(x_1 \vee \ldots \vee x_k, \infty)$$

Combinatorial Auctions as Weighted Partial MaxSAT

Goods

Agents	1	2	3	4
1	10 x_1	15 x_2	-	-
2	-	5 x_3	10 x_4	20 x_5
3	15 x_6	-	-	10 x_7

$(x_1, 10)$ $\quad (\overline{x}_1 \vee \overline{x}_6, \infty)$

$(x_2, 15)$ $\quad (\overline{x}_2 \vee \overline{x}_3, \infty)$

$(x_3, 5)$ $\quad (\overline{x}_5 \vee \overline{x}_7, \infty)$

$(x_4, 10)$

$(x_5, 20)$ $\quad (x_1 \vee x_2, \infty)$

$(x_6, 15)$ $\quad (x_3 \vee x_4 \vee x_5, \infty)$

$(x_7, 10)$ $\quad (x_6 \vee x_7, \infty)$

Combinatorial Auctions as Weighted Partial MaxSAT

Goods

Agents	1	2	3	4
1	10 x_1	15 x_2	-	-
2	-	5 x_3	10 x_4	20 x_5
3	15 x_6	-	-	10 x_7

$$\left.\begin{array}{l} (x_1, 10) \\ (x_2, 15) \\ (x_3, 5) \\ (x_4, 10) \\ (x_5, 20) \\ (x_6, 15) \\ (x_7, 10) \end{array}\right\} \text{Cost of the bids}$$

$(\overline{x}_1 \vee \overline{x}_6, \infty)$

$(\overline{x}_2 \vee \overline{x}_3, \infty)$

$(\overline{x}_5 \vee \overline{x}_7, \infty)$

$(x_1 \vee x_2, \infty)$

$(x_3 \vee x_4 \vee x_5, \infty)$

$(x_6 \vee x_7, \infty)$

Combinatorial Auctions as Weighted Partial MaxSAT

Goods

Agents	1	2	3	4
1	10 x_1	15 x_2	-	-
2	-	5 x_3	10 x_4	20 x_5
3	15 x_6	-	-	10 x_7

$(x_1, 10)$ $\quad (\overline{x}_1 \vee \overline{x}_6, \infty)$

$(x_2, 15)$ $\quad (\overline{x}_2 \vee \overline{x}_3, \infty)$

$(x_3, 5)$ $\quad (\overline{x}_5 \vee \overline{x}_7, \infty)$

Compatibility of bids

$(x_4, 10)$

$(x_5, 20)$ $\quad (x_1 \vee x_2, \infty)$

$(x_6, 15)$ $\quad (x_3 \vee x_4 \vee x_5, \infty)$

$(x_7, 10)$ $\quad (x_6 \vee x_7, \infty)$

Combinatorial Auctions as Weighted Partial MaxSAT

Goods

Agents	1	2	3	4
1	10 (x_1)	15 (x_2)	-	-
2	-	5 (x_3)	10 (x_4)	20 (x_5)
3	15 (x_6)	-	-	10 (x_7)

$(x_1, 10)$ $(\overline{x}_1 \vee \overline{x}_6, \infty)$

$(x_2, 15)$ $(\overline{x}_2 \vee \overline{x}_3, \infty)$

$(x_3, 5)$ $(\overline{x}_5 \vee \overline{x}_7, \infty)$

$(x_4, 10)$

$(x_5, 20)$ $(x_1 \vee x_2, \infty)$

$(x_6, 15)$ $(x_3 \vee x_4 \vee x_5, \infty)$

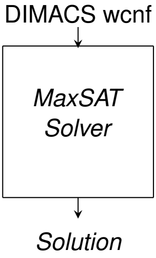
$(x_7, 10)$ $(x_6 \vee x_7, \infty)$

Minimum winning bids

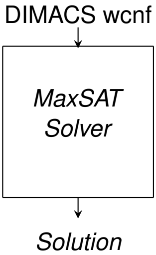
		Goods			
		1	2	3	4
Agents	1	10 (x_1)	15 (x_2)	-	-
	2	-	5 (x_3)	10 (x_4)	20 (x_5)
	3	15 (x_6)	-	-	10 (x_7)

$$(x_1, 10) \qquad (\overline{x}_1 \vee \overline{x}_6, \infty)$$
$$(x_2, 15) \qquad (\overline{x}_2 \vee \overline{x}_3, \infty)$$
$$(x_3, 5) \qquad (\overline{x}_5 \vee \overline{x}_7, \infty)$$
$$(x_4, 10)$$
$$(x_5, 20) \qquad (x_1 \vee x_2, \infty)$$
$$(x_6, 15) \qquad (x_3 \vee x_4 \vee x_5, \infty)$$
$$(x_7, 10) \qquad (x_6 \vee x_7, \infty)$$

Using MaxSAT solvers



Using MaxSAT solvers



Auction formulation

$$
\begin{array}{l}
(x_1, 10) \\
(x_2, 15) \\
(x_3, 5) \\
(x_4, 10) \\
(x_5, 20) \\
(x_6, 15) \\
(x_7, 10) \\
(\overline{x}_1 \vee \overline{x}_6, \infty) \\
(\overline{x}_2 \vee \overline{x}_3, \infty) \\
(\overline{x}_5 \vee \overline{x}_7, \infty) \\
(x_1 \vee x_2, \infty) \\
(x_3 \vee x_4 \vee x_5, \infty) \\
(x_6 \vee x_7, \infty)
\end{array}
$$

Using MaxSAT solvers

DIMACS wcnf

↓

MaxSAT Solver

↓

Solution

Auction formulation

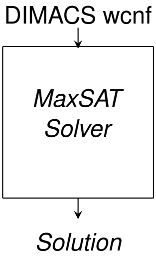
$(x_1, 10)$
$(x_2, 15)$
$(x_3, 5)$
$(x_4, 10)$
$(x_5, 20)$
$(x_6, 15)$
$(x_7, 10)$
$(\overline{x}_1 \vee \overline{x}_6, \infty)$
$(\overline{x}_2 \vee \overline{x}_3, \infty)$
$(\overline{x}_5 \vee \overline{x}_7, \infty)$
$(x_1 \vee x_2, \infty)$
$(x_3 \vee x_4 \vee x_5, \infty)$
$(x_6 \vee x_7, \infty)$

Formula in DIMACS wcnf

p wcnf 7 13 86

10	1	0		
15	2	0		
5	3	0		
10	4	0		
20	5	0		
15	6	0		
10	7	0		
86	-1	-6	0	
86	-2	-3	0	
86	-5	-7	0	
86	1	2	0	
86	3	4	5	0
86	6	7	0	

Using MaxSAT solvers



Execution output

o 25
s OPTIMUM FOUND
v -1 2 -3 4 5 6 -7

Auction formulation

$(x_1, 10)$
$(x_2, 15)$
$(x_3, 5)$
$(x_4, 10)$
$(x_5, 20)$
$(x_6, 15)$
$(x_7, 10)$
$(\overline{x}_1 \lor \overline{x}_6, \infty)$
$(\overline{x}_2 \lor \overline{x}_3, \infty)$
$(\overline{x}_5 \lor \overline{x}_7, \infty)$
$(x_1 \lor x_2, \infty)$
$(x_3 \lor x_4 \lor x_5, \infty)$
$(x_6 \lor x_7, \infty)$

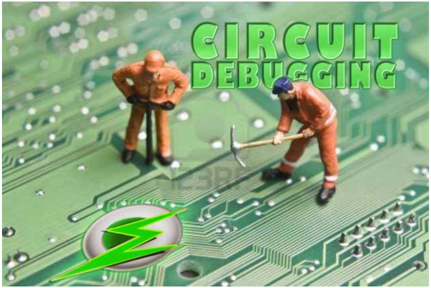
Formula in DIMACS wcnf

p wcnf 7 13 86

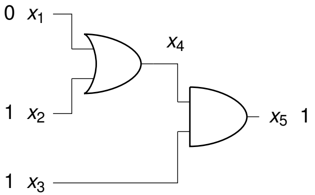
10	1	0		
15	2	0		
5	3	0		
10	4	0		
20	5	0		
15	6	0		
10	7	0		
86	-1	-6	0	
86	-2	-3	0	
86	-5	-7	0	
86	1	2	0	
86	3	4	5	0
86	6	7	0	

Instance: an erroneous circuit, an input stimulus and the corresponding correct output

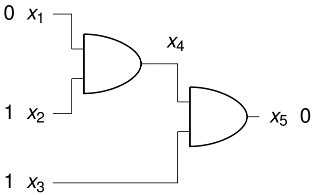
Question: maximum number of satisfied circuit gates?



correct circuit:



erroneous circuit:



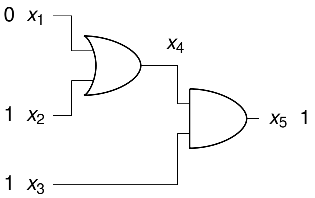
We use as Boolean variables the inputs, outputs and gates

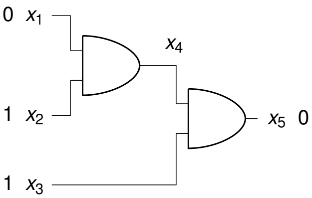
Standard formulation

- Soft: CNF representation of the erroneous circuit as soft clauses of weight 1
- Hard: for each input and output force its value as a hard clause

Circuit Design Debugging as Partial MaxSAT

correct circuit: erroneous circuit:

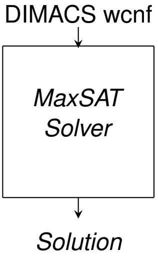




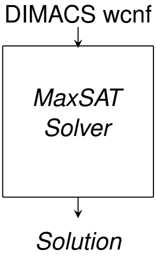
Circuit formulation

$$
\begin{array}{l}
(\overline{x}_4 \vee x_1, 1) \\
(\overline{x}_4 \vee x_2, 1) \\
(x_4 \vee \overline{x}_1 \vee \overline{x}_2, 1) \\
(\overline{x}_5 \vee x_4, 1) \\
(\overline{x}_5 \vee x_3, 1) \\
(x_5 \vee \overline{x}_4 \vee \overline{x}_3, 1) \\
(\overline{x}_1, \infty) \\
(x_2, \infty) \\
(x_3, \infty) \\
(x_5, \infty)
\end{array}
$$

Circuit Design Debugging as Partial MaxSAT



Circuit Design Debugging as Partial MaxSAT



Circuit formulation

$(\overline{x}_4 \vee x_1, 1)$
$(\overline{x}_4 \vee x_2, 1)$
$(x_4 \vee \overline{x}_1 \vee \overline{x}_2, 1)$
$(\overline{x}_5 \vee x_4, 1)$
$(\overline{x}_5 \vee x_3, 1)$
$(x_5 \vee \overline{x}_4 \vee \overline{x}_3, 1)$
$(\overline{x}_1, \infty)$
(x_2, ∞)
(x_3, ∞)
(x_5, ∞)

Circuit Design Debugging as Partial MaxSAT

DIMACS wcnf

↓

MaxSAT Solver

↓

Solution

Circuit formulation

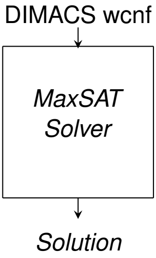
$$
\begin{array}{l}
(\overline{x}_4 \vee x_1, 1) \\
(\overline{x}_4 \vee x_2, 1) \\
(x_4 \vee \overline{x}_1 \vee \overline{x}_2, 1) \\
(\overline{x}_5 \vee x_4, 1) \\
(\overline{x}_5 \vee x_3, 1) \\
(x_5 \vee \overline{x}_4 \vee \overline{x}_3, 1) \\
(\overline{x}_1, \infty) \\
(x_2, \infty) \\
(x_3, \infty) \\
(x_5, \infty)
\end{array}
$$

Formula in DIMACS wcnf

p wcnf 5 10 7

1	-4	1	0	
1	-4	2	0	
1	4	-1	-2	0
1	-5	4	0	
1	-5	3	0	
1	5	-4	-3	0
7	-1	0		
7	2	0		
7	3	0		
7	5	0		

Circuit Design Debugging as Partial MaxSAT



Execution output

o 1
s OPTIMUM FOUND
v -1 2 3 -4 5 0

Circuit formulation

$$(\overline{x}_4 \vee x_1, 1)$$
$$(\overline{x}_4 \vee x_2, 1)$$
$$(x_4 \vee \overline{x}_1 \vee \overline{x}_2, 1)$$
$$(\overline{x}_5 \vee x_4, 1)$$
$$(\overline{x}_5 \vee x_3, 1)$$
$$(x_5 \vee \overline{x}_4 \vee \overline{x}_3, 1)$$
$$(\overline{x}_1, \infty)$$
$$(x_2, \infty)$$
$$(x_3, \infty)$$
$$(x_5, \infty)$$

Formula in DIMACS wcnf

p wcnf 5 10 7

1	-4	1	0	
1	-4	2	0	
1	4	-1	-2	0
1	-5	4	0	
1	-5	3	0	
1	5	-4	-3	0
7	-1	0		
7	2	0		
7	3	0		
7	5	0		

Software Package Upgrades

Instance: set of packages $p_i \in \{p_1, \ldots, p_n\}$,

where for each package $p_i = (p_i, D, C)$ we have:

a set D of dependencies (required p_j) for installing p_i
(a dependency $d \in D$ is a disjunction of packages)

a set C of conflicts (disallowed p_j) for installing p_i
(a conflict is a single package)

Question: maximum number of packages to be installed?

Softw. Package Upgrade as Partial MaxSAT

We define the following Boolean variables:

- $X = \{x_1, \ldots, x_n\}$,
  where $x_i = 1$ means the package p_i will be installed.

Standard formulation

For each package $p_i = (p_i, D, C)$:

- Soft: $(x_i, 1)$
- Hard (*dependencies*): $\bigwedge_{d \in D} (\overline{x}_i \vee d, \infty)$
  where $d = \bigvee_{p_j \in d} x_j$
- Hard (*conflicts*): $\bigwedge_{p_j \in C} (\overline{x}_i \vee \overline{x}_j, \infty)$

Softw. Package Upgrade as Partial MaxSAT

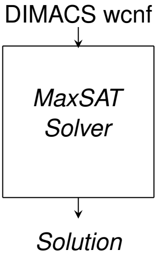
Package constraints

$$(p_1, \{p_2 \vee p_3\}, \{p_4\})$$
$$(p_2, \{p_3\}, \{p_4\})$$
$$(p_3, \{p_2\}, \emptyset)$$
$$(p_4, \{p_2, p_3\}, \emptyset)$$

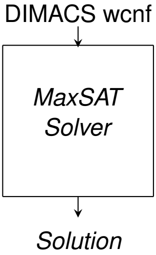
Partial MaxSAT Formulation

$$(x_1, 1)$$
$$(x_2, 1)$$
$$(x_3, 1)$$
$$(x_4, 1)$$
$$(\overline{x}_1 \vee x_2 \vee x_3, \infty)$$
$$(\overline{x}_1 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_2 \vee x_3, \infty)$$
$$(\overline{x}_2 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_3 \vee x_2, \infty)$$
$$(\overline{x}_4 \vee x_2, \infty)$$
$$(\overline{x}_4 \vee x_3, \infty)$$

Softw. Package Upgrade as Partial MaxSAT



Softw. Package Upgrade as Partial MaxSAT



Circuit formulation

$$
\begin{array}{l}
(x_1, 1) \\
(x_2, 1) \\
(x_3, 1) \\
(x_4, 1) \\
(\overline{x}_1 \vee x_2 \vee x_3, \infty) \\
(\overline{x}_1 \vee \overline{x}_4, \infty) \\
(\overline{x}_2 \vee x_3, \infty) \\
(\overline{x}_2 \vee \overline{x}_4, \infty) \\
(\overline{x}_3 \vee x_2, \infty) \\
(\overline{x}_4 \vee x_2, \infty) \\
(\overline{x}_4 \vee x_3, \infty)
\end{array}
$$

Softw. Package Upgrade as Partial MaxSAT

DIMACS wcnf

↓

MaxSAT Solver

↓

Solution

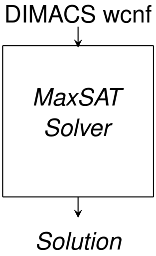
Circuit formulation

$$
\begin{array}{l}
(x_1, 1) \\
(x_2, 1) \\
(x_3, 1) \\
(x_4, 1) \\
(\overline{x}_1 \vee x_2 \vee x_3, \infty) \\
(\overline{x}_1 \vee \overline{x}_4, \infty) \\
(\overline{x}_2 \vee x_3, \infty) \\
(\overline{x}_2 \vee \overline{x}_4, \infty) \\
(\overline{x}_3 \vee x_2, \infty) \\
(\overline{x}_4 \vee x_2, \infty) \\
(\overline{x}_4 \vee x_3, \infty)
\end{array}
$$

Formula in DIMACS wcnf

```
p wcnf 4 11 5
1  1  0
1  2  0
1  3  0
1  4  0
5 -1  2  3  0
5 -1 -4  0
5 -2  3  0
5 -2 -4  0
5 -3  2  0
5 -4  2  0
5 -4  3  0
```

Softw. Package Upgrade as Partial MaxSAT



Execution output

o 1
s OPTIMUM FOUND
v 1 2 3 -4 0

Circuit formulation

$$(x_1, 1)$$
$$(x_2, 1)$$
$$(x_3, 1)$$
$$(x_4, 1)$$
$$(\overline{x}_1 \vee x_2 \vee x_3, \infty)$$
$$(\overline{x}_1 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_2 \vee x_3, \infty)$$
$$(\overline{x}_2 \vee \overline{x}_4, \infty)$$
$$(\overline{x}_3 \vee x_2, \infty)$$
$$(\overline{x}_4 \vee x_2, \infty)$$
$$(\overline{x}_4 \vee x_3, \infty)$$

Formula in DIMACS wcnf

p wcnf 4 11 5

1	1	0		
1	2	0		
1	3	0		
1	4	0		
5	-1	2	3	0
5	-1	-4	0	
5	-2	3	0	
5	-2	-4	0	
5	-3	2	0	
5	-4	2	0	
5	-4	3	0	

(k_1,k_2)-Weighted Partial MaxSAT

Let (k_1,k_2)-Weighted Partial MaxSAT, where $k_1, k_2 \geq 1$ are integers, be the special case of Weighted Partial MaxSAT in which all soft clauses have k_1 literals and all hard clauses have k_2 literals

To (1,3)-Weighted Partial MaxSAT

Be φ a MaxSAT formula, for each $(C_i, w_i) \in \varphi$:

- Replace (C_i, w_i) by $(\overline{b}_i, w_i)$
  where b_i is a reification variable
- Add $(CNF(\overline{C}_i \leftrightarrow b_i), \infty)$
  where $CNF(\Gamma)$ is the Conjunctive Normal Form of Γ

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

To (1,3)-Weighted Partial MaxSAT

Be φ a MaxSAT formula, for each $(C_i, w_i) \in \varphi$:

- Replace (C_i, w_i) by $(\overline{b}_i, w_i)$
  where b_i is a reification variable
- Add $(CNF(\overline{C}_i \leftrightarrow b_i), \infty)$
  where $CNF(\Gamma)$ is the Conjunctive Normal Form of Γ

Reified Constraints

Reified constraints reflect the validity of a constraint C into a Boolean variable

Above, clause C_i is reified into the *reification* variable b_i

b_i is true $\leftrightarrow$ C_i is false

To (1,3)-Weighted Partial MaxSAT

Be $\varphi = \{(C_1, 1), \ldots, (C_m, 1)\}$ a MaxSAT formula.

The translation to (1,3)-Weighted Partial MaxSAT is:

$$
\begin{array}{ll}
(C_1, 1) & (\overline{b}_1, 1) \\
\ldots & \ldots \\
(C_m, 1) & (\overline{b}_m, 1) \\
 & (CNF(\overline{C}_1 \leftrightarrow b_1), \infty) \\
 & \ldots \\
 & (CNF(\overline{C}_m \leftrightarrow b_m), \infty)
\end{array}
$$

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

To (1,3)-Weighted Partial MaxSAT

Be $\varphi = \{(C_1, 1), \ldots, (C_m, 1)\}$ a MaxSAT formula.

The translation to (1,3)-Weighted Partial MaxSAT is:

$$
\begin{array}{ll}
(C_1, 1) & (\overline{b}_1, 1) \\
\ldots & \ldots \\
(C_m, 1) & (\overline{b}_m, 1) \\
 & (CNF(\overline{C}_1 \rightarrow b_1), \infty) \\
 & \ldots \\
 & (CNF(\overline{C}_m \rightarrow b_m), \infty)
\end{array}
$$

We can skip $\overline{C}_i \leftarrow b_i$ since we are maximizing $\overline{b}_i$

To (1,3)-Weighted Partial MaxSAT

Be $\varphi = \{(C_1, 1), \ldots, (C_m, 1)\}$ a MaxSAT formula.

The translation to (1,3)-Weighted Partial MaxSAT is:

$$
\begin{array}{ll}
(C_1, 1) & (\overline{b}_1, 1) \\
\ldots & \ldots \\
(C_m, 1) & (\overline{b}_m, 1) \\
 & (C_1 \vee b_1, \infty) \\
 & \ldots \\
 & (C_m \vee b_m, \infty)
\end{array}
$$

We can skip $\overline{C}_i \leftarrow b_i$ since we are maximizing $\overline{b}_i$

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

Example

Let be $C_i = (x_1 \vee x_2, 1)$

Replace C_i by:

$$(\overline{b}_i, 1)$$
$$(CNF((\overline{x}_1 \wedge \overline{x}_2) \leftrightarrow b_i), \infty)$$

Note: $\overline{C}_i = (\overline{x}_1 \wedge \overline{x}_2)$

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

Example

Let be $C_i = (x_1 \vee x_2, 1)$

Replace C_i by:

$$
\begin{array}{l}
(\overline{b}_i, 1) \\
(CNF((\overline{x}_1 \wedge \overline{x}_2) \rightarrow b_i), \infty) \\
(CNF((\overline{x}_1 \wedge \overline{x}_2) \leftarrow b_i), \infty)
\end{array}
$$

Note: $\overline{C_i} = (\overline{x}_1 \wedge \overline{x}_2)$

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

Example

Let be $C_i = (x_1 \vee x_2, 1)$

Replace C_i by:

$$\begin{array}{l} (\overline{b}_i, 1) \\ (x_1 \vee x_2 \vee b_i, \infty) \\ (CNF((\overline{x}_1 \wedge \overline{x}_2) \leftarrow b_i), \infty) \end{array}$$

Note: $\overline{C}_i = (\overline{x}_1 \wedge \overline{x}_2)$

Example

Let be $C_i = (x_1 \vee x_2, 1)$

Replace C_i by:

$$\begin{array}{l}(\overline{b}_i, 1)\\(x_1 \vee x_2 \vee b_i, \infty)\\(\overline{x}_1 \vee \overline{b}_i, \infty)\\(\overline{x}_2 \vee \overline{b}_i, \infty)\end{array}$$

Note: $\overline{C}_i = (\overline{x}_1 \wedge \overline{x}_2)$

Weighted MaxSAT to (1,3)-Weighted Partial MaxSAT

Example

Let be $C_i = (x_1 \vee x_2, 1)$

Replace C_i by:

$$\begin{array}{l}(\overline{b}_i, 1)\\(x_1 \vee x_2 \vee b_i, \infty)\end{array}$$

We can skip $\overline{C}_i \leftarrow b_i$ since we are maximizing $\overline{b}_i$
Note: $\overline{C}_i = (\overline{x}_1 \wedge \overline{x}_2)$

Max2SAT is NP-Hard

Definition

A problem is *NP-hard* if all problems in NP are reducible to it in polynomial-time. In other words, an optimization problem is NP-hard if the existence of a polynomial time algorithm solving it (to optimality) implies P=NP

Proof idea:

Reduce a 3SAT instance φ to a Max2SAT instance φ' such that φ is satisfiable $\leftrightarrow$ $cost(\varphi') = 3 \cdot m$,

where m is the number of clauses in φ

Then, a polynomial algorithm for MAX2SAT would provide a polynomial algorithm for 3SAT

Max2SAT is NP-Hard

To prove NP-Hardness, we reduce 3SAT to Max2SAT:

Reduction of 3SAT φ to Max2SAT φ' [Papadimitriou, 1994]

For each clause $(x_1 \vee x_2 \vee x_3) \in \varphi$:

- Introduce a new variable b
- Replace $(x_1 \vee x_2 \vee x_3)$ by the following 10 soft clauses:

Block I	$(x_1), (x_2), (x_3), (b)$
Block II	$(\overline{x}_1 \vee \overline{x}_2), (\overline{x}_1 \vee \overline{x}_3), (\overline{x}_2 \vee \overline{x}_3)$
Block III	$(x_1 \vee \overline{b}), (x_2 \vee \overline{b}), (x_3 \vee \overline{b})$

φ is satisfiable $\leftrightarrow$ $cost(\varphi') = 3 \cdot m$,

where m is the number of clauses in φ

Max2SAT is NP-Hard

3SAT to Max2SAT: gadget construction

Replace $C = (x_1 \vee x_2 \vee x_3)$ by φ'_C:

Block I	$(x_1), (x_2), (x_3), (b)$
Block II	$(\overline{x}_1 \vee \overline{x}_2), (\overline{x}_1 \vee \overline{x}_3), (\overline{x}_2 \vee \overline{x}_3)$
Block III	$(x_1 \vee \overline{b}), (x_2 \vee \overline{b}), (x_3 \vee \overline{b})$

$(x_1 \vee x_2 \vee x_3)$ is false $\leftrightarrow$ $cost(\varphi'_C) = 4$

$(x_1 \vee x_2 \vee x_3)$ is true $\leftrightarrow$ $cost(\varphi'_C) = 3$

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

International MaxSAT Evaluation

International MaxSAT Evaluation 2013

Argelich, J., Li, C., Manyà, F., Planes, J.: http://www.maxsat.udl.cat/

Instances:

Random, crafted, industrial

Categories:

MaxSAT (378, 167, 55)

Partial MaxSAT(177, 270, 504)

Weighted MaxSAT(160, 116, _)

Weighted Partial MaxSAT(120, 372, 226)

Solvers:

Complete (Exact): Branch and bound, SAT-based, ILP

Incomplete: Local search, Exact solvers reporting *lb* and *ub*

Summary of MaxSAT Evaluations

Branch and Bound Sat-Based ILP Portfolio

Previous MaxSAT Evaluations (1800s, 0.5GB)

	2008			2009			2012		
	random	crafted	industrial	random	crafted	industrial	random	crafted	industrial
MS	IncMz LB-Sat	IncMz LB-Sat	IncMz LB-Sat	IBMz IBLMz	IBMz IBLMz	msu wbo	akms_ls iut_rr_rv	akms_ls iut_rr_rv	wbo1.6 wpm1
WMS	Clone IncMz	Clone IncMz		IncMz IBCWz	IBCWz IBCLWz		akms_ls iut_rr_rv	akms_ls akms	
PMS	Clone IncMz	Clone IncMz	Clone IncMz	IBCWz Wz-2.5	Wz-2.5 Wz-1.6	pm2 sat4j	akms_ls akms	qms akms_ls	qms_g2 pwbo2.0
WPMS	Clone IncMz	Clone IncMz	Clone IncMz	Wz-2.5 Wz-1.6	IncWz Clone	sat4j IncWz	akms_ls akms	wpm1 shinms	pwbo2.1 wpm1

Current MSE (1800s, 3.5GB)

	2013		
	random	crafted	industrial
MS	MSz13f ISAC+	ISAC+ MSz13f	pmifum WPM11
WMS	ckm-s ISAC+	WMSz+ WMSz+	
PMS	ISAC+ WMSZ9	ISAC+ ILP	ISAC+ QMS2
WPMS	ISAC+ WMSz9	MaxHS ISAC	WPM1 ISAC+

Summary of MaxSAT Evaluations

Branch and Bound | Sat-Based | ILP | Portfolio

Previous MaxSAT Evaluations (1800s, 0.5GB)

	2008			2009			2012		
	random	crafted	industrial	random	crafted	industrial	random	crafted	industrial
MS	IncMz LB-Sat	IncMz LB-Sat	IncMz LB-Sat	IBMz IBLMz	IBMz IBLMz	msu wbo	akms_ls iut_rr_rv	akms_ls iut_rr_rv	wbo1.6 wpm1
WMS	Clone IncMz	Clone IncMz		IncMz IBCWz	IBCWz IBCLWz		akms_ls iut_rr_rv	akms_ls akms	
PMS	Clone IncMz	Clone IncMz	Clone IncMz	IBCWz Wz-2.5	Wz-2.5 Wz-1.6	pm2 sat4j	akms_ls akms	qms akms_ls	qms_g2 pwbo2.0
WPMS	Clone IncMz	Clone IncMz	Clone IncMz	Wz-2.5 Wz-1.6	IncWz Clone	sat4j IncWz	akms_ls akms	wpm1 shinms	pwbo2.1 wpm1

Current MSE (1800s, 3.5GB)

	2013		
	random	crafted	industrial
MS	MSz13f ckm-s	MSz13f ckm-s	pmifum WPM11
WMS	ckm-s MSz13f	WMSz+ WMSz+	
PMS	WMSZ9 WMSZ+	ILP scip-ms	QMS2 MSUC
WPMS	WMSz9 WMSz+	MaxHS ILP	WPM1 WPM2

Exact MaxSAT solvers

- Branch and bound based
  - extend DPLL or CDCL SAT algorithms
  - apply incomplete MaxSAT resolution rules
  - use underestimation techniques
  - hard to extend new SAT solvers
  - best performance on random instances
- Satisfiability testing based (SAT-based)
  - solve a sequence of SAT formulas
  - exploit unsatisfiable cores and satisfying assignments
  - introduce PseudoBoolean (PB) linear constraints
  - easy to incorporate new SAT solvers
  - best performance on industrial instances

The MaxSAT problem on a formula φ can be solved through the resolution of a sequence of SAT formulas in the following way:

Let φ_k be SAT formula which is satisfiable iff φ has an interpretation with a cost less than or equal to k:

If optimum cost of φ is k', then the SAT formulas φ_k, for $k \geq k'$ are satisfiable, while for $k < k'$ are unsatisfiable.

SAT-based MaxSAT solvers

The MaxSAT problem on a formula φ can be solved through the resolution of a sequence of SAT formulas in the following way:

Let φ_k be SAT formula which is satisfiable iff φ has an interpretation with a cost less than or equal to k:

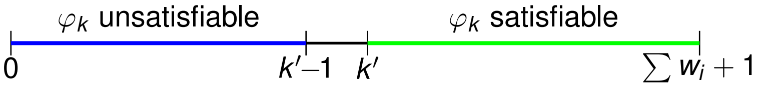
If optimum cost of φ is k', then the SAT formulas φ_k, for $k \geq k'$ are satisfiable, while for $k < k'$ are unsatisfiable.

SAT-based MaxSAT solvers

The MaxSAT problem on a formula φ can be solved through the resolution of a sequence of SAT formulas in the following way:

Let φ_k be SAT formula which is satisfiable iff φ has an interpretation with a cost less than or equal to k:

If optimum cost of φ is k', then the SAT formulas φ_k, for $k \geq k'$ are satisfiable, while for $k < k'$ are unsatisfiable.



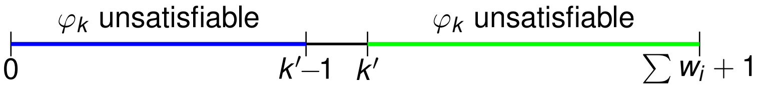
How do we build φ_k?

Let $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

- Replace every soft clause (C_i, w_i) by $\overline{C_i} \rightarrow b_i \equiv C_i \vee b_i$
- Replace every hard clause (C_{m+j}, ∞) by C_{m+j}
- Add the conversion to CNF of $\sum w_i \cdot b_i \leq k$

$$\varphi_k = \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}, CNF(\sum b_i * w_i \leq k)\}$$

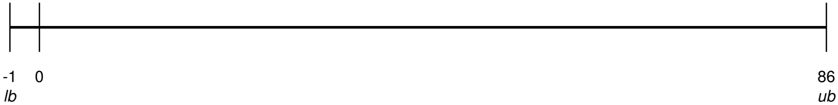
SAT-based MaxSAT solvers



In which order do we *check* the φ_k's?

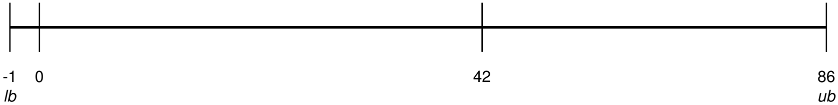
- From 0 to k' (unsatisfiable side)?
- From $\sum w_i + 1$ to $k' - 1$ (satisfiable side)?
- Alternating both?

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ lower bound
$ub := \sum w_i + 1$ upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

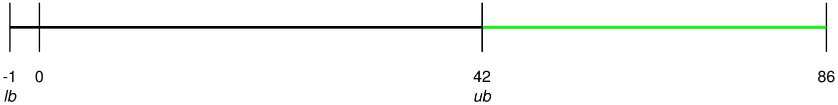
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

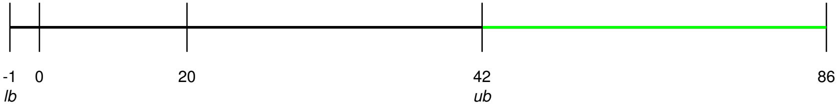
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ lower bound
$ub := \sum w_i + 1$ upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

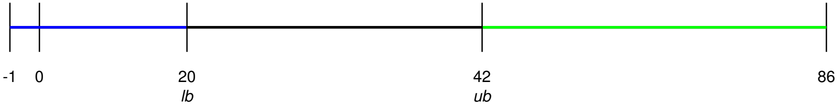
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

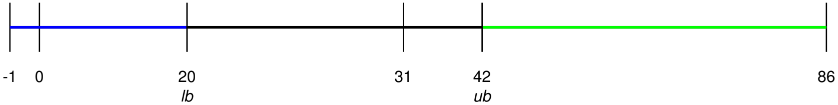
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

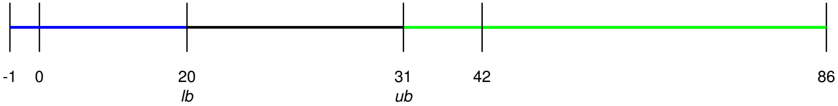
return ub

Binary search for MaxSAT



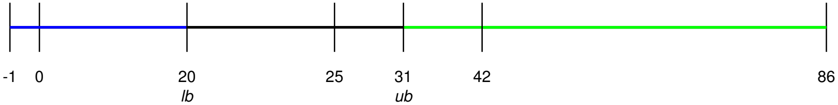
input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ — lower bound
$ub := \sum w_i + 1$ — upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ — SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ — lower bound
$ub := \sum w_i + 1$ — upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ — SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ lower bound

$ub := \sum w_i + 1$ upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

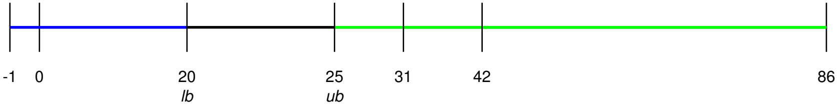
  $st := SAT(\varphi_k)$ SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

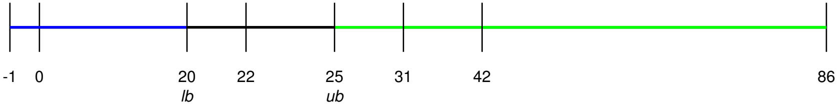
return ub

Binary search for MaxSAT



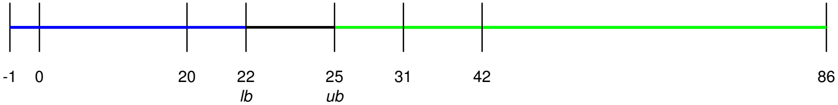
input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ lower bound
$ub := \sum w_i + 1$ upper bound
while $lb + 1 < ub$ **do**
    $k := \lfloor (lb + ub)/2 \rfloor$ middle point
    $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
    $st := SAT(\varphi_k)$ SAT call
    if $st = sat$ **then** $ub := k$
    if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



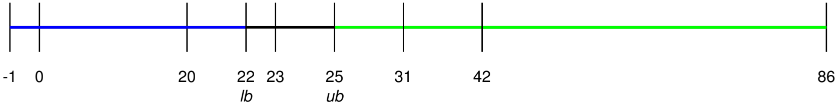
input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ — lower bound
$ub := \sum w_i + 1$ — upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ — SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ lower bound
$ub := \sum w_i + 1$ upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

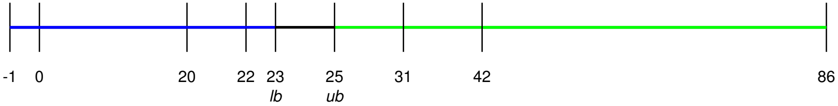
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

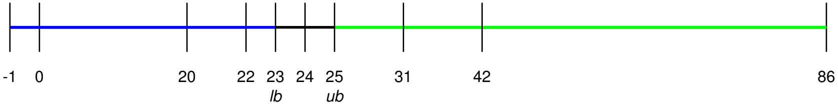
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

return ub

Binary search for MaxSAT



input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$

$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$

$lb := -1$ — lower bound

$ub := \sum w_i + 1$ — upper bound

while $lb + 1 < ub$ **do**

  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point

  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$

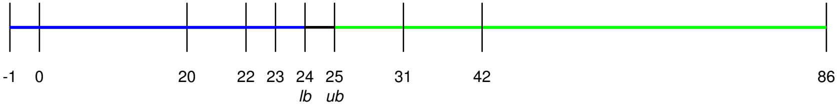
  $st := SAT(\varphi_k)$ — SAT call

  if $st = sat$ **then** $ub := k$

  if $st = unsat$ **then** $lb := k$

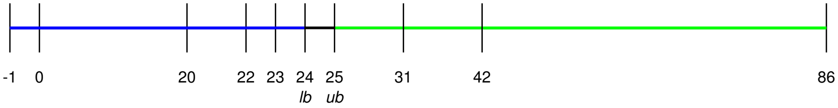
return ub

Binary search for MaxSAT



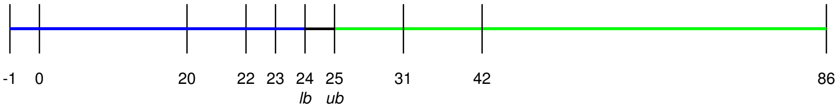
input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$lb := -1$ — lower bound
$ub := \sum w_i + 1$ — upper bound
while $lb + 1 < ub$ **do**
  $k := \lfloor (lb + ub)/2 \rfloor$ — middle point
  $\varphi_k = \varphi' \cup CNF(\sum w_i \cdot b_i \leq k)$
  $st := SAT(\varphi_k)$ — SAT call
  if $st = sat$ **then** $ub := k$
  if $st = unsat$ **then** $lb := k$
return ub

Binary search for MaxSAT



How can we improve it ?

Binary search for MaxSAT



How can we improve it ?

First of all, check the latest SAT competition results and select an efficient SAT solver!

Key points of efficient SAT-based MaxSAT solvers

- How to search for the optimum?

  unsat≪sat, unsat≫sat, unsat≫≪sat

- How to exploit SAT solvers' output?

  unsatisfiable cores: $\uparrow lb$

  satisfying assignments: $\downarrow ub$

  keep learned clauses: incremental mode

  activity scores?

- How to produce simpler PB constraints?

  length, coefficients, independent term, etc.

- How to manage PB constraints?

  SAT, PB, ILP, SMT, Lazy decomposition, etc.

Cardinality and Pseudo-Boolean Constraints

- Cardinality Constraints:

$$\sum_{x_i \in X} x_i \triangleright k$$

where X is a set of propositional atoms, k is an integer positive constant and $\triangleright$ is one of the operators of $\{=, <, \leq, >, \geq\}$

- Pseudo-Boolean (PB) Constraints:

$$\sum_{x_i \in X} c_i \cdot x_i \triangleright k$$

where c_i is an integer constant, k a positive integer constant and $\triangleright$ is one of the operators of $\{=, <, \leq, >, \geq\}$

Cardinality and Pseudo-Boolean Constraints

- Example: Cumulative resource constraints
  - Some tasks $\{1, 2, \cdots, n\}$ must be done.
  - Tasks require some (limited) resources.
  - Variable $x_{i,t}$ is true if task i is active at time t.
  - Constraint: There are no more active tasks than machines.

$$x_{1,t} + x_{2,t} + \cdots + x_{n,t} \leqslant 20$$

  - Constraint: We are not exceeding the number of workers.

$$3x_{1,t} + 4x_{2,t} + \cdots + 10x_{n,t} \leqslant 50$$

- Encode the constraint into SAT: decompose the constraint into an equisatisfiable set of clauses.

Arc-consistent Encodings

Definition

Given a constraint C, an equisatisfiable SAT encoding with clause set S, and a partial assignment A, we say that:

S is *arc-consistent*, if whenever x_i is true (false) in every extension of A satisfying C, then unit propagating A on S sets x_i to true (false).

Enforcing arc-consistency by unit propagation in this way has an important positive impact on the practical efficiency of SAT solvers.

Summary of Cardinality Encodings

Encoding	Reference	Vars	Clauses	Consist.	GAC
Warners	[Warners, 1998]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	NO	NO
Totalizers	[Bailleux and Boufkhad, 2003]	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2)$	YES	YES
Parallel Counters	[Sinz, 2005]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	NO	NO
Sequential Counters	[Sinz, 2005]	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	YES	YES
BDD	[Bailleux et al., 2006]	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	YES	YES
Sorting Networks	[Eén and Sörensson, 2006a]	$\mathcal{O}(n \log^2 n)$	$\mathcal{O}(n \log^2 n)$	YES	YES
Cardinality Networks	[Asín et al., 2011]	$\mathcal{O}(n \log^2 k)$	$\mathcal{O}(n \log^2 k)$	YES	YES
Pairwise Card. Networks	[Codish and Zazon-Ivry, 2010]	$\mathcal{O}(n \log^2 k)$	$\mathcal{O}(n \log^2 k)$	YES	YES

Table: Summary comparing the different cardinality encodings.

Translating PB Constraints into SAT

- Translate the PB constraint into a BDD, which can be treated as a circuit of ITEs (if-then-else gates) and translated into clauses by the Tseitin transformation. The resulting encoding is arc-consistent but its size is exponential in the worst case
- Translate the PB constraint into a network of adders. The approach is similar to the what is used for Data Multipliers to sum up the partial products. The size of the translation is $O(n)$, however the resulting encoding is not arc-consistent
- Translate the PB constraint into a network of sorters.
  The size of the translation is $O(n \cdot \log^2 n)$, and although it is not yet arc-consistent it is closer than the translation through adders

Parameter n is the total number of digits in all the coefficients.

Example

Consider the following PB constraint:

$$x_1 + x_2 + 2 \cdot x_3 + 3 \cdot x_3 + 3 \cdot x_4 + 3 \cdot x_5 + 3 \cdot x_6 + 7 \cdot x_7 \geq 8$$

Its translation to PB format is:

```
+1*x1 +1*x2 +2*x3 +3*x3 +3*x4 +3*x5 + 3*x6 +7*x7 >= 8;
```

Lets run minisat+, options:

- -cb: use BDDs.
- -ca: use Adders.
- -cs: use Sorters.
- -cnf="file.cnf”: export SAT formula to file.cnf

```
./minisat+ file.pb -cs -cnf="file.cnf"
```

Summary of PseudoBoolean Encodings

Encoding	Reference	Clauses	Consist.	GAC
Warners	[Warners, 1998]	$\mathcal{O}(n \log a_{max})$	NO	NO
Non-reduced BDD	[Bailleux et al., 2006]	Exponential	YES	YES
ROBDD	[Eén and Sörensson, 2006a]	Exponential	YES	YES
Adders	[Eén and Sörensson, 2006a]	$\mathcal{O}(\sum \log a_i)$	NO	NO
Sorting Networks	[Eén and Sörensson, 2006a]	$\mathcal{O}((\sum \log a_i) \log^2(\sum \log a_i))$	YES	NO
Watch Dog (WD)	[Bailleux et al., 2009]	$\mathcal{O}(n^2 \log n \log a_{max})$	YES	NO
Gen. Arc-cons. WD	[Bailleux et al., 2009]	$\mathcal{O}(n^3 \log n \log a_{max})$	YES	YES
Power-of-Two BDD	[Abío et al., 2012]	$\mathcal{O}(n^2 \log a_{max})$	YES	NO
Gen Arc-cons Power-of-Two BDD	[Abío et al., 2012]	$\mathcal{O}(n^3 \log a_{max})$	YES	YES

Table: Summary comparing the different PB encodings.

- **SAT Modulo Theories (SMT)**

  Determine satisfiability of a first order formula F w.r.t. a background theory T:

  $$A \wedge (B \vee x + 3 < y) \wedge x \geq y$$

  Boolean model:
  $A = \mathit{true}, B = \mathit{true}, (x + 3 < y) = \mathit{false}, (x \geq y) = \mathit{true}$
  if T is the theory of Linear Integer Arithmetic.
- Most common **SMT-solvers** use **SAT-solver + T-solver**
- SMT well suited for CSP solving: see fzn2smt in Minizinc challenge

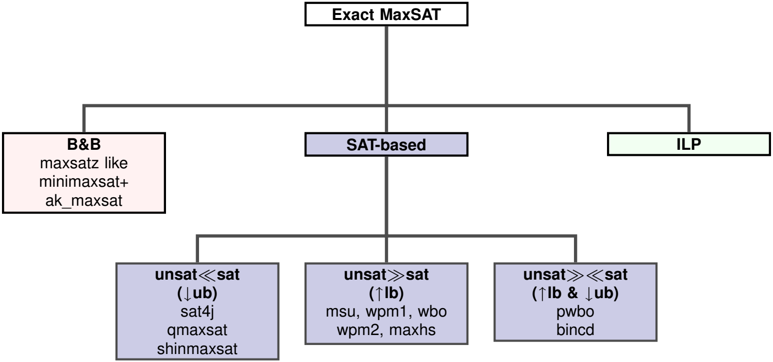
- **SAT Modulo Theories (SMT)**

  Determine satisfiability of a first order formula F w.r.t. a background theory T:

  $$A \wedge (B \vee x + 3 < y) \wedge x \geq y$$

  Boolean model:
  $A = true, B = true, (x + 3 < y) = false, (x \geq y) = true$
  if T is the theory of Linear Integer Arithmetic.
- Most common **SMT-solvers** use **SAT-solver + T-solver**
- SMT well suited for CSP solving: see fzn2smt in Minizinc challenge

Taxonomy of Modern Exact MaxSAT solvers



Working example: Pigeon Hole Problem (PHP)

Consider the pigeon-hole formula with 5 pigeons and one hole

Variables: x_i means that pigeon i goes to the only hole

Soft clauses: w_i is the cost of pigeon i out of the hole

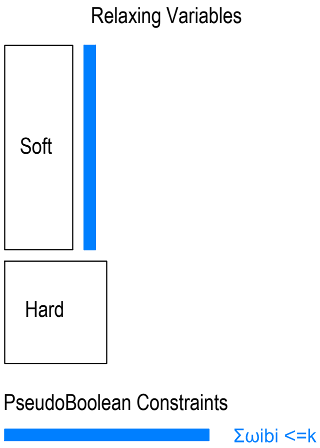
Hard clauses: at most one pigeon into the hole

$$
\begin{aligned}
\varphi = \{ & (x_1 \quad, w_1), \\
& (x_2 \quad, w_2), \\
& (x_3 \quad, w_3), \\
& (x_4 \quad, w_4), \\
& (x_5 \quad, w_5) \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)
\end{aligned}
$$

- Originally applied in minisat+ [Eén and Sörensson, 2006b]
- SAT4J algorithm [Berre, 2006] version for MaxSAT
- Explores from sat to unsat
- Exploits satisfying assignments (model-guided)
- One relaxing b_i variable per soft clause
- b_i variables added initially
- Adds PB constraints of the form $\sum w_i \cdot b_i \leq k$
- Current solvers:

  sat4java [Berre, 2006]
  qmaxsat [Koshimura et al., 2012]
  shinmaxsat [Honjyo and Tanjo, 2012]

SAT4JAVA


Relaxing Variables
Soft
Hard
PseudoBoolean Constraints
Σωibi <=k

Example: SAT4J on PHP^5_1

$$\begin{aligned}\varphi_{17} = \{ & (x_1 \vee b_1, 1),\\ & (x_2 \vee b_2, 3),\\ & (x_3 \vee b_3, 3),\\ & (x_4 \vee b_4, 5),\\ & (x_5 \vee b_5, 5)\} \cup\\ & CNF(\sum x_i \leq 1, \infty) \cup\\ & CNF(1 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 5 \cdot b_4 + 5 \cdot b_5 \leq 17, \infty) \quad \blacksquare\end{aligned}$$

$\mathcal{I}(b_1, b_2, b_3, b_4, b_5) = (1, 0, 1, 1, 1)$;
$\mathcal{I}(\varphi_{17}) = 14$;

Example: SAT4J on PHP^5_1

$$
\begin{aligned}
\varphi_{13} = \{\ & (x_1 \vee b_1\ , 1), \\
& (x_2 \vee b_2\ , 3), \\
& (x_3 \vee b_3\ , 3), \\
& (x_4 \vee b_4\ , 5), \\
& (x_5 \vee b_5\ , 5)\} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(1 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 5 \cdot b_4 + 5 \cdot b_5 \leq 13\ , \infty) \ \blacksquare
\end{aligned}
$$

$\mathcal{I}(b_1, b_2, b_3, b_4, b_5) = (1, 1, 1, 0, 1);$
$\mathcal{I}(\varphi_{13}) = 12;$

Example: SAT4J on PHP^5_1

$$
\begin{aligned}
\varphi_{11} = \{ & (x_1 \vee b_1, 1),\\
& (x_2 \vee b_2, 3),\\
& (x_3 \vee b_3, 3),\\
& (x_4 \vee b_4, 5),\\
& (x_5 \vee b_5, 5)\} \cup\\
& CNF(\sum x_i \leq 1, \infty) \cup\\
& CNF(1 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 5 \cdot b_4 + 5 \cdot b_5 \leq 11, \infty)
\end{aligned}
$$

UNSAT

$\mathcal{I}(\varphi) = 12$

input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$ub := \sum w_i + 1$
while true do
 $\varphi_{ub} = \varphi' \cup CNF(\sum w_i \cdot b_i \leq ub - 1)$ — φ_{ub} construction
 $(st, \mathcal{I}) := SAT(\varphi_{ub})$ — SAT call
 if $st = sat$ **then** $ub := \mathcal{I}(\varphi)$ — ub refinement
 if $st = unsat$ **then return** ub

input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
$\varphi' := \{C_1 \vee b_1, \ldots, C_m \vee b_m, C_{m+1}, \ldots, C_{m+m'}\}$
$ub := \sum w_i + 1$
while true do
  $\varphi_{ub} = \varphi' \cup CNF(\sum w_i \cdot b_i \leq ub - 1)$ φ_{ub} construction
  $(st, \mathcal{I}) := SAT(\varphi_{ub})$ SAT call
  if $st = sat$ **then** $ub := \mathcal{I}(\varphi)$ ub refinement
  if $st = unsat$ **then return** ub

Example on Combinatorial Auctions: 86, 60, 50, 40, 25.

- Originally designed for solving Partial MaxSAT [Fu and Malik, 2006]
- Explores from unsat to sat
- Exploits unsatisfiable cores
- Can have more than one b_i variable per clause
- b_i variables added on demand
- Adds PB constraints of the form $\sum b_i = 1$
- Current solvers:

  MSU1.X [Marques-Silva and Planes, 2007a]
  WPM1 [Ansotegui et al., 2009]
  WBO [Manquinho et al., 2009]

unsat≫sat: Fu and Malik algorithm

Unsatisfiable Core (UC)

Given an unsatisfiable SAT formula φ, an *unsatisfiable core* φ_c is a subset of clauses $\varphi_c \subseteq \varphi$ that is also unsatisfiable

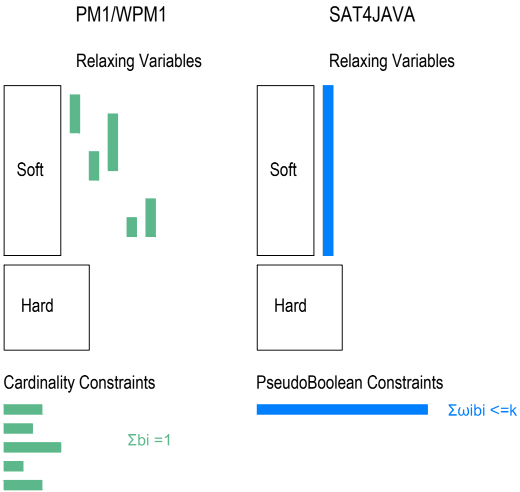
Minimal UCs

A *minimal UC* is an UC such that any proper subset of it is satisfiable

SAT solvers return an UC when the answer is unsat

The UC returned is not guaranteed to be minimal

unsat≫sat: Fu and Malik algorithm



Example: Fu and Malik on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{ \; & (x_1 \quad, 1), \\
& (x_2 \quad, 1), \\
& (x_3 \quad, 1), \\
& (x_4 \quad, 1), \\
& (x_5 \quad, 1) \; \} \cup \\
& CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

Example: Fu and Malik on PHP_1^5

$$\begin{aligned}
\varphi_0 = \{ & (x_1 \quad, 1), \blacksquare \\
& (x_2 \quad, 1), \blacksquare \\
& (x_3 \quad, 1), \\
& (x_4 \quad, 1), \\
& (x_5 \quad, 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \blacksquare
\end{aligned}$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$

Example: Fu and Malik on PHP^5_1

$$
\begin{aligned}
\varphi_1 = \{\ & (x_1 \vee b_1^1\ , 1),\ \blacksquare \\
& (x_2 \vee b_2^1\ , 1),\ \blacksquare \\
& (x_3\ , 1), \\
& (x_4\ , 1), \\
& (x_5\ , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)\ \blacksquare
\end{aligned}
$$

Soft clauses of the core are relaxed

A cardinality constraint is added as hard clause

$\mathcal{I}(\varphi) \geq 1$

Example: Fu and Malik on PHP_1^5

$$
\begin{aligned}
\varphi_1 = \{\ & (x_1 \vee b_1^1\ , 1), \\
& (x_2 \vee b_2^1\ , 1), \\
& (x_3\ , 1), \blacksquare \\
& (x_4\ , 1), \blacksquare \\
& (x_5\ , 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)\ \blacksquare \\
& CNF(b_1^1 + b_2^1 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 1$

Example: Fu and Malik on PHP$^5_1$

$$
\begin{aligned}
\varphi_2 = \{\ & (x_1 \vee b_1^1 \quad\quad , 1), \\
& (x_2 \vee b_2^1 \quad\quad , 1), \\
& (x_3 \vee \quad\quad b_3^2, 1), \blacksquare \\
& (x_4 \vee \quad\quad b_4^2, 1), \blacksquare \\
& (x_5 \quad\quad\quad\quad , 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) \geq 2$

Example: Fu and Malik on PHP_1^5

$$
\begin{aligned}
\varphi_2 = \{\ & (x_1 \vee b_1^1 \quad , 1), \\
& (x_2 \vee b_2^1 \quad , 1), \\
& (x_3 \vee \quad b_3^2 , 1), \\
& (x_4 \vee \quad b_4^2 , 1), \\
& (x_5 \qquad\quad , 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 2$

$$
\begin{aligned}
\varphi_3 = \{\ & (x_1 \vee b_1^1 \vee \quad\quad b_1^3\ , 1), \blacksquare \\
& (x_2 \vee b_2^1 \vee \quad\quad b_2^3\ , 1), \blacksquare \\
& (x_3 \vee \quad\quad b_3^2 \vee b_3^3\ , 1), \blacksquare \\
& (x_4 \vee \quad\quad b_4^2 \vee b_4^3\ , 1), \blacksquare \\
& (x_5 \quad\quad\quad\quad\quad\quad , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) \geq 3$

Example: Fu and Malik on PHP_1^5

$$
\begin{aligned}
\varphi_3 = \{ & (x_1 \vee b_1^1 \vee \quad b_1^3, 1), \blacksquare \\
& (x_2 \vee b_2^1 \vee \quad b_2^3, 1), \blacksquare \\
& (x_3 \vee \quad b_3^2 \vee b_3^3, 1), \blacksquare \\
& (x_4 \vee \quad b_4^2 \vee b_4^3, 1), \blacksquare \\
& (x_5 \quad , 1) \ \blacksquare \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \ \blacksquare \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \ \blacksquare \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \ \blacksquare \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) > 3$

$$
\begin{aligned}
\varphi_4 = \{ & (x_1 \vee b_1^1 \vee b_1^3 \vee b_1^4, 1), \blacksquare \\
& (x_2 \vee b_2^1 \vee b_2^3 \vee b_2^4, 1), \blacksquare \\
& (x_3 \vee b_3^2 \vee b_3^3 \vee b_3^4, 1), \blacksquare \\
& (x_4 \vee b_4^2 \vee b_4^3 \vee b_4^4, 1), \blacksquare \\
& (x_5 \vee b_5^4, 1) \ \blacksquare \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \\
& CNF(b_1^4 + b_2^4 + b_3^4 + b_4^4 + b_5^4 = 1, \infty) \ \blacksquare
\end{aligned}
$$

SAT

$\mathcal{I}(\varphi) = 4$

Fu and Malik algorithm

Input: $\varphi = \{(C_1, 1), \ldots, (C_m, 1), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $\infty, \emptyset$;
$s := 0$;
while *true* **do**
    $(st, \varphi_c, \mathcal{I}) := SAT(\{C_i \mid (C_i, w_i) \in \varphi\})$;
    if $st = \text{SAT}$ **then return** $\mathcal{I}(\varphi), \mathcal{I}$;
    $s := s + 1$;
    $A := \emptyset$; /* Indexes of the core */
    foreach $C_i \in \varphi_c$ **do**
        if $w_i \neq \infty$ **then** /* If the clause is soft */
            $b_i^s :=$ new_variable( );
            $\varphi := \varphi \setminus \{(C_i, 1)\} \cup \{(C_i \vee b_i^s, 1)\}$; /* Relax clause */
            $A := A \cup \{i\}$;
        end
    end
    $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A} b_i^s = 1)\}$
end

Lets look again to φ_3 without clause $(x_5, 1)$

$$
\begin{array}{llll}
x_1 \vee & \boxed{b_1^1} \vee & & b_1^3 \\
x_2 \vee & b_2^1 \vee & & \boxed{b_2^3} \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}
$$

$\boxed{b_1^1} + b_2^1 = 1$

$\boxed{b_3^2} + b_4^2 = 1$

$b_1^3 + \boxed{b_2^3} + b_3^3 + b_4^3 = 1$

$$
\begin{array}{llll}
x_1 \vee & b_1^1 \vee & & \boxed{b_1^3} \\
x_2 \vee & \boxed{b_2^1} \vee & & b_2^3 \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}
$$

$b_1^1 + \boxed{b_2^1} = 1$

$\boxed{b_3^2} + b_4^2 = 1$

$\boxed{b_1^3} + b_2^3 + b_3^3 + b_4^3 = 1$

Breaking Symmetries in the Fu and Malik algorithm

Lets look again to φ_3 without clause $(x_5, 1)$

The formula is satisfiable, but it has 8 models instead of 4

$$\begin{array}{llll}
x_1 \vee & \boxed{b_1^1} \vee & & b_1^3 \\
x_2 \vee & b_2^1 \vee & & \boxed{b_2^3} \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3 \\
\multicolumn{4}{l}{\boxed{b_1^1} + b_2^1 = 1} \\
\multicolumn{4}{l}{\boxed{b_3^2} + b_4^2 = 1} \\
\multicolumn{4}{l}{b_1^3 + \boxed{b_2^3} + b_3^3 + b_4^3 = 1}
\end{array}
\qquad
\begin{array}{llll}
x_1 \vee & b_1^1 \vee & & \boxed{b_1^3} \\
x_2 \vee & \boxed{b_2^1} \vee & & b_2^3 \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3 \\
\multicolumn{4}{l}{b_1^1 + \boxed{b_2^1} = 1} \\
\multicolumn{4}{l}{\boxed{b_3^2} + b_4^2 = 1} \\
\multicolumn{4}{l}{\boxed{b_1^3} + b_2^3 + b_3^3 + b_4^3 = 1}
\end{array}$$

Breaking Symmetries in the Fu and Malik algorithm

The two previous models are related by the permutation

$$b_1^1 \leftrightarrow b_2^1, b_1^3 \leftrightarrow b_2^3$$

The existence of so many *partial* models makes the task of showing unsatisfiability of the formula (including x_5) much harder.

$$\begin{array}{llll}
x_1 \vee & \boxed{b_1^1} \vee & & b_1^3 \\
x_2 \vee & b_2^1 \vee & & \boxed{b_2^3} \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}$$

$$\boxed{b_1^1} + b_2^1 = 1$$

$$\boxed{b_3^2} + b_4^2 = 1$$

$$b_1^3 + \boxed{b_2^3} + b_3^3 + b_4^3 = 1$$

$$\begin{array}{llll}
x_1 \vee & b_1^1 \vee & & \boxed{b_1^3} \\
x_2 \vee & \boxed{b_2^1} \vee & & b_2^3 \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}$$

$$b_1^1 + \boxed{b_2^1} = 1$$

$$\boxed{b_3^2} + b_4^2 = 1$$

$$\boxed{b_1^3} + b_2^3 + b_3^3 + b_4^3 = 1$$

Breaking Symmetries in the Fu and Malik algorithm

The two previous models are related by the permutation

$$b_1^1 \leftrightarrow b_2^1, b_1^3 \leftrightarrow b_2^3$$

The existence of so many *partial* models makes the task of showing unsatisfiability of the formula (including x_5) much harder.

$$\begin{array}{llll}
x_1 \vee & \boxed{b_1^1} \vee & & b_1^3 \\
x_2 \vee & b_2^1 \vee & & \boxed{b_2^3} \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}$$

$$\boxed{b_1^1} + b_2^1 = 1$$
$$\boxed{b_3^2} + b_4^2 = 1$$
$$b_1^3 + \boxed{b_2^3} + b_3^3 + b_4^3 = 1$$

$$\begin{array}{llll}
x_1 \vee & b_1^1 \vee & & \boxed{b_1^3} \\
x_2 \vee & \boxed{b_2^1} \vee & & b_2^3 \\
x_3 \vee & & \boxed{b_3^2} \vee & b_3^3 \\
\boxed{x_4} \vee & & b_4^2 \vee & b_4^3
\end{array}$$

$$b_1^1 + \boxed{b_2^1} = 1$$
$$\boxed{b_3^2} + b_4^2 = 1$$
$$\boxed{b_1^3} + b_2^3 + b_3^3 + b_4^3 = 1$$

Breaking Symmetries in the Fu and Malik algorithm

The two previous models are related by the permutation

$$b_1^1 \leftrightarrow b_2^1, b_1^3 \leftrightarrow b_2^3$$

The existence of so many *partial* models makes the task of showing unsatisfiability of the formula (including x_5) much harder.

After finding the third unsatisfiable core, we break symmetries by adding to φ_3:

$$\begin{array}{l} b_1^3 \rightarrow \overline{b}_2^1 \\ b_3^3 \rightarrow \overline{b}_4^2 \end{array}$$

- Weighted Fu and Malik introduced in:

  WPM1 [Ansotegui et al., 2009]
  WBO [Manquinho et al., 2009]

  Idea:

- Be w_{min}, the minimum weight into the unsat core
- Replace soft clauses in the unsat core by,

  a relaxed copy with w_{min} and cardinality

  a copy with $w_i - w_{min}$

- Increase the current cost by w_{min}

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{\; & (x_1 \quad , 1), \\
& (x_2 \quad , 3), \\
& (x_3 \quad , 3), \\
& (x_5 \quad , 5), \\
& (x_5 \quad , 5) \;\} \cup \\
& CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{\ & (x_1\ \ , 1),\ \blacksquare \\
& (x_2\ \ , 3),\ \blacksquare \\
& (x_3\ \ , 3), \\
& (x_4\ \ , 5), \\
& (x_5\ \ , 5)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty)\ \blacksquare
\end{aligned}
$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$

Example: WPM1 on PHP^5_1

$$\varphi_0 = \{ \; (x_1 \quad, 1), \; \blacksquare$$
$$(x_2 \quad, 3), \; \blacksquare$$
$$(x_3 \quad, 3),$$
$$(x_4 \quad, 5),$$
$$(x_5 \quad, 5) \; \} \cup$$
$$CNF(\sum x_i \le 1, \infty) \; \blacksquare$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$

$$\varphi_1 = \{ \; (x_1 \vee \; b^1_1 \quad, 1), \; \blacksquare$$
$$(x_2 \vee \; b^1_2 \quad, 1), \; \blacksquare$$
$$(x_3 \quad, 3),$$
$$(x_4 \quad, 5),$$
$$(x_5 \quad, 5),$$
$$(x_2 \quad, 2) \; \blacksquare \} \cup$$
$$CNF(\sum x_i \le 1, \infty) \cup$$
$$CNF(b^1_1 + b^1_2 = 1, \infty) \; \blacksquare$$

copy with $w_i - w_{min}$

copy with w_{min}

$\mathcal{I}(\varphi) \ge 1$

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_1 = \{ & (x_1 \vee b_1^1 \quad, 1), \\
& (x_2 \vee b_2^1 \quad, 1), \\
& (x_3 \qquad\quad, 3), \blacksquare \\
& (x_4 \qquad\quad, 5), \blacksquare \\
& (x_5 \qquad\quad, 5), \\
& (x_2 \qquad\quad, 2) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \ \blacksquare \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 1$

Example: WPM1 on PHP^5_1

$$\begin{aligned}
\varphi_1 = \{\ & (x_1 \vee b_1^1 &, 1),\\
& (x_2 \vee b_2^1 &, 1),\\
& (x_3 &, 3), \blacksquare\\
& (x_4 &, 5), \blacksquare\\
& (x_5 &, 5),\\
& (x_2 &, 2)\ \} \cup\\
& CNF(\textstyle\sum x_i \leq 1, \infty)\ \blacksquare \} \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty)
\end{aligned}$$

$$\begin{aligned}
\varphi_3 = \{\ & (x_1 \vee b_1^1 &, 1),\\
& (x_2 \vee b_2^1 &, 1),\\
& (x_3 \vee b_3^2 &, 3), \blacksquare\\
& (x_4 \vee b_4^2 &, 3), \blacksquare\\
& (x_5 &, 5),\\
& (x_2 &, 2),\\
& (x_4 &, 2)\ \blacksquare \} \cup\\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty)\\
& CNF(b_3^2 + b_4^2 = 1, \infty)\ \blacksquare
\end{aligned}$$

$\mathcal{I}(\varphi) > 1$

$\mathcal{I}(\varphi) \geq 3$

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_3 = \{\ & (x_1 \vee b_1^1 \quad , 1), \\
& (x_2 \vee b_2^1 \quad , 1), \\
& (x_3 \vee \quad b_3^2 , 3), \\
& (x_4 \vee \quad b_4^2 , 3), \\
& (x_5 \quad , 5), \\
& (x_2 \quad , 2), \blacksquare \\
& (x_4 \quad , 2) \ \blacksquare \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \ \blacksquare \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 3$

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_3 = \{\ & (x_1 \vee b_1^1 \quad , 1),\\
& (x_2 \vee b_2^1 \quad , 1),\\
& (x_3 \vee \quad b_3^2 , 3),\\
& (x_4 \vee \quad b_4^2 , 3),\\
& (x_5 \quad\quad , 5),\\
& (x_2 \quad\quad , 2),\\
& (x_4 \quad\quad , 2) \quad \} \cup\\
& CNF(\textstyle\sum x_i \leq 1, \infty) \quad \} \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty)\\
& CNF(b_3^2 + b_4^2 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 3$

$$
\begin{aligned}
\varphi_5 = \{\ & (x_1 \vee b_1^1 \quad\quad , 1),\\
& (x_2 \vee b_2^1 \quad\quad , 1),\\
& (x_3 \vee \quad b_3^2 \quad , 3),\\
& (x_4 \vee \quad b_4^2 \quad , 3),\\
& (x_5 \quad\quad\quad , 5),\\
& (x_2 \vee \quad\quad b_6^3 , 2),\\
& (x_4 \vee \quad\quad b_7^3 , 2) \quad \} \cup\\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup\\
& CNF(b_1^1 + b_2^1 = 1, \infty)\\
& CNF(b_3^2 + b_4^2 = 1, \infty)\\
& CNF(b_6^3 + b_7^3 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) \geq 5$

Example: WPM1 on PHP^5_1

$$
\begin{aligned}
\varphi_5 = \{\ & (x_1 \vee b_1^1 \qquad\qquad , 1), \blacksquare \\
& (x_2 \vee b_2^1 \qquad\qquad , 1), \blacksquare \\
& (x_3 \vee \quad b_3^2 \qquad , 3), \\
& (x_4 \vee \quad b_4^2 \qquad , 3), \\
& (x_5 \qquad\qquad\qquad , 5), \blacksquare \\
& (x_2 \vee \qquad\quad b_6^3 , 2), \\
& (x_4 \vee \qquad\quad b_7^3 , 2)\} \cup \\
& CNF(\sum x_i \leq 1, \infty)\ \blacksquare\} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)\ \blacksquare \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_6^3 + b_7^3 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 5$

Example: WPM1 on PHP_1^5

$$\varphi_5 = \{ (x_1 \vee b_1^1, 1), \blacksquare$$
$$(x_2 \vee b_2^1, 1), \blacksquare$$
$$(x_3 \vee b_3^2, 3),$$
$$(x_4 \vee b_4^2, 3),$$
$$(x_5, 5), \blacksquare$$
$$(x_2 \vee b_6^3, 2),$$
$$(x_4 \vee b_7^3, 2)\} \cup$$
$$CNF(\sum x_i \leq 1, \infty) \ \blacksquare\} \cup$$
$$CNF(b_1^1 + b_2^1 = 1, \infty) \ \blacksquare$$
$$CNF(b_3^2 + b_4^2 = 1, \infty)$$
$$CNF(b_6^3 + b_7^3 = 1, \infty)$$

$\mathcal{I}(\varphi) > 5$

$$\varphi_6 = \{ (x_1 \vee b_1^1 \vee b_1^4, 1), \blacksquare$$
$$(x_2 \vee b_2^1 \vee b_2^4, 1), \blacksquare$$
$$(x_3 \vee b_3^2, 3),$$
$$(x_4 \vee b_4^3, 3),$$
$$(x_5 \vee b_5^4, 1), \blacksquare$$
$$(x_2 \vee b_6^3, 2),$$
$$(x_4 \vee b_7^3, 2),$$
$$(x_5, 4) \ \blacksquare\} \cup$$
$$CNF(\sum x_i \leq 1, \infty) \cup$$
$$CNF(b_1^1 + b_2^1 = 1, \infty)$$
$$CNF(b_3^2 + b_4^2 = 1, \infty)$$
$$CNF(b_6^3 + b_7^3 = 1, \infty)$$
$$CNF(b_1^4 + b_2^4 + b_5^4 = 1, \infty) \ \blacksquare$$

$\mathcal{I}(\varphi) \geq 6$

Example: WPM1 on PHP^5_1

$$
\begin{array}{lllllllll}
\varphi_{12} = \{ & (x_1 \vee & b_1^1 \vee & & & b_1^4 \vee & & & b_1^7 & , 1), \\
& (x_2 \vee & b_2^1 \vee & & & b_2^4 \vee & & & b_2^7 & , 1), \\
& (x_3 \vee & & b_3^2 \vee & & & b_3^5 \vee & & b_3^7 & , 1), \\
& (x_4 \vee & & b_4^2 \vee & & & b_4^5 \vee & & b_4^7 & , 1), \\
& (x_5 \vee & & & & b_5^4 \vee & & & b_5^7 & , 1), \\
& (x_2 \vee & & & b_6^3 \vee & & & b_6^6 & & , 1), \\
& (x_4 \vee & & & b_7^3 \vee & & & b_7^6 & & , 1), \\
& (x_5 \vee & & & & & b_8^5 \vee & & b_8^7 & , 1), \\
& (x_5 \vee & & & & & & b_9^6 & & , 1), \\
& (x_2 \vee & & & b_6^3 \vee & & & & b_{10}^7 & , 1), \\
& (x_4 \vee & & & b_7^3 \vee & & & & b_{11}^7 & , 1), \\
& (x_3 \vee & & b_3^2 \vee & & & b_3^5 & & & , 2), \\
& (x_4 \vee & & b_4^2 \vee & & & b_4^5 & & & , 2), \\
& (x_5 \vee & & & & & b_8^5 & & & , 2) \ \} \cup
\end{array}
$$

$$
\begin{array}{l}
CNF(\sum x_i \leq 1, \infty) \cup \\
CNF(b_1^1 + b_2^1 = 1, \infty) \\
CNF(b_3^2 + b_4^2 = 1, \infty) \\
CNF(b_6^3 + b_7^3 = 1, \infty) \\
CNF(b_1^4 + b_2^4 + b_5^4 = 1, \infty) \\
CNF(b_3^5 + b_4^5 + b_8^5 = 1, \infty) \\
CNF(b_6^6 + b_7^6 + b_9^6 = 1, \infty) \\
CNF(b_1^7 + b_2^7 + b_3^7 + b_4^7 + b_5^7 + b_8^7 + b_{10}^7 + b_{11}^7 = 1, \infty)
\end{array}
$$

SAT

$\mathcal{I}(\varphi) = 12$

WPM1 algorithm

Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$;
$s := 0$; /* Counter of cores */
while *true* **do**
  $(st, \varphi_c, \mathcal{I}) := SAT(\{C_i \mid (C_i, w_i) \in \varphi)$;
  if $st = \text{SAT}$ **then return** $\mathcal{I}(\varphi), \mathcal{I}$;
  else
    $s := s + 1$;
    $A := \emptyset$; /* Indexes of the core */
    $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$; /* Minimum weight */
    foreach $C_i \in \varphi_c$ **do**
      if $w_i \neq \infty$ **then**
        $b_i^s :=$ new_variable();
        $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b_i^s, w_{min})\}$
        $A := A \cup \{i\}$
      end
    $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A} b_i^s = 1)\}$; /* Cardinal. */
    end
  end
end

Stratified approach for the WPM1 algorithm

Problem in $wpm1_{2009}$:

- the number of iterations depends on w_{min}
- SAT solvers have no notion of weights
- SAT solvers can return not minimal unsat cores

Stratified approach (applied in $wpm1_{2011}$):

- force SAT solver to focus on clauses with higher weights
- only clauses with $w_i > w_{max}$ are sent to the solver
- when SAT solver returns sat, w_{max} is decreased
- copies with $w_i - w_{min}$ are rescheduled

Stratified Approach in the WPM1 algorithm

$$\begin{aligned}\varphi_0 = \{\ & (x_5 \quad , 5),\\ & (x_4 \quad , 5),\\ & (x_3 \quad , 3),\\ & (x_2 \quad , 3),\\ & (x_1 \quad , 1)\ \} \cup\\ & CNF(\textstyle\sum x_i \leq 1, \infty)\end{aligned}$$

Stratified Approach in the WPM1 algorithm

$$
\begin{aligned}
\varphi_0 = \{ \ & (x_5 \quad , 5), \\
& (x_4 \quad , 5), \\
& (x_3 \quad , 3), \\
& (x_2 \quad , 3), \\
& (x_1 \quad , 1) \ \} \cup \\
& CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$

Stratified Approach in the WPM1 algorithm

$$\varphi_0 = \{\ (x_5\ , 5),\ \blacksquare$$
$$(x_4\ , 5),\ \blacksquare$$
$$(x_3\ , 3),$$
$$(x_2\ , 3),$$
$$(x_1\ , 1)\ \} \cup$$
$$CNF(\sum x_i \leq 1, \infty)\ \blacksquare$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$

$$\varphi_5 = \{\ (x_5 \vee\ b_1^1\ , 5),\ \blacksquare$$
$$(x_4 \vee\ b_2^1\ , 5),\ \blacksquare$$
$$(x_3\ , 3),$$
$$(x_2\ , 3),$$
$$(x_1\ , 1)\} \cup$$
$$CNF(\sum x_i \leq 1, \infty) \cup$$
$$CNF(b_1^1 + b_2^1 = 1, \infty)\ \blacksquare$$

copy with w_{min}

$\mathcal{I}(\varphi) \geq 5$

Stratified Approach in the WPM1 algorithm

$$
\begin{aligned}
\varphi_5 = \{\ & (x_5 \vee b_1^1 \quad, 5), \\
& (x_4 \vee b_2^1 \quad, 5), \\
& (x_3 \qquad\quad, 3), \\
& (x_2 \qquad\quad, 3), \\
& (x_1 \qquad\quad, 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)\ \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)
\end{aligned}
$$

$\mathcal{I}(\varphi) > 5$

Stratified Approach in the WPM1 algorithm

$$\varphi_5 = \{\ (x_5 \vee b_1^1\ , 5),\ (x_4 \vee b_2^1\ , 5),\ (x_3\ , 3),\ \blacksquare\ (x_2\ , 3),\ \blacksquare\ (x_1\ , 1)\ \} \cup CNF(\sum x_i \leq 1, \infty)\ \blacksquare \} \cup CNF(b_1^1 + b_2^1 = 1, \infty)$$

$\mathcal{I}(\varphi) > 5$

$$\varphi_8 = \{\ (x_5 \vee b_1^1 \vee\ , 5),\ (x_4 \vee b_2^1 \vee\ , 5),\ (x_3 \vee\ b_3^2\ , 3),\ \blacksquare\ (x_2 \vee\ b_4^2\ , 3),\ \blacksquare\ (x_1\ , 1), \} \cup CNF(\sum x_i \leq 1, \infty) \cup CNF(b_1^1 + b_2^1 = 1, \infty)\ CNF(b_3^2 + b_4^2 = 1, \infty)\ \blacksquare$$

$\mathcal{I}(\varphi) \geq 8$

Stratified Approach in the WPM1 algorithm

$$
\begin{aligned}
\varphi_8 = \{\ & (x_5 \vee b_1^1 \quad , 5), \blacksquare \\
& (x_4 \vee b_2^1 \quad , 5), \blacksquare \\
& (x_3 \vee \quad b_3^2 , 3), \blacksquare \\
& (x_2 \vee \quad b_4^2 , 3), \blacksquare \\
& (x_1 \qquad , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty)\ \blacksquare \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)\ \blacksquare \\
& CNF(b_3^2 + b_4^2 = 1, \infty)\ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) > 8$

Stratified Approach in the WPM1 algorithm

$$
\begin{aligned}
\varphi_8 = \{\ & (x_5 \vee b_1^1\ ,\ 5),\ \blacksquare \\
& (x_4 \vee b_2^1\ ,\ 5),\ \blacksquare \\
& (x_3 \vee b_3^2\ ,\ 3),\ \blacksquare \\
& (x_2 \vee b_4^2\ ,\ 3),\ \blacksquare \\
& (x_1\ ,\ 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)\ \blacksquare \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty)\ \blacksquare \\
& CNF(b_3^2 + b_4^2 = 1, \infty)\ \blacksquare
\end{aligned}
$$

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1^1 \vee b_1^3,\ 3),\ \blacksquare \\
& (x_4 \vee b_2^1 \vee b_2^3,\ 3),\ \blacksquare \\
& (x_3 \vee b_3^2 \vee b_3^3,\ 3),\ \blacksquare \\
& (x_2 \vee b_4^2 \vee b_4^3,\ 3),\ \blacksquare \\
& (x_5 \vee b_1^1\ ,\ 2),\ \blacksquare \\
& (x_4 \vee b_2^1\ ,\ 2),\ \blacksquare \\
& (x_1\ ,\ 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_2^3 + b_3^4 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)\ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) > 8$

copy with $w_i - w_{min}$

$\mathcal{I}(\varphi) \geq 11$

Stratified Approach in the WPM1 algorithm

$$\begin{array}{rlllll}
\varphi_{11} = \{ & (x_5 \vee & b_1^1 \vee & & b_1^3, & 3), \\
& (x_4 \vee & b_2^1 \vee & & b_2^3, & 3), \\
& (x_3 \vee & & b_3^2 \vee & b_3^3, & 3), \\
& (x_2 \vee & & b_4^2 \vee & b_4^3, & 3), \\
& (x_5 \vee & b_1^1 & & , & 2), \\
& (x_4 \vee & b_2^1 & & , & 2), \\
& (x_1 & & & , & 1) \} \cup \\
& \multicolumn{5}{l}{CNF(\sum x_i \leq 1, \infty) \} \cup} \\
& \multicolumn{5}{l}{CNF(b_1^1 + b_2^1 = 1, \infty)} \\
& \multicolumn{5}{l}{CNF(b_3^2 + b_4^2 = 1, \infty)} \\
& \multicolumn{5}{l}{CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)}
\end{array}$$

$\mathcal{I}(\varphi) > 11$

Stratified Approach in the WPM1 algorithm

$$\varphi_{11} = \{ (x_5 \vee b_1^1 \vee b_1^3, 3), (x_4 \vee b_2^1 \vee b_2^3, 3), (x_3 \vee b_3^2 \vee b_3^3, 3), (x_2 \vee b_4^2 \vee b_4^3, 3), (x_5 \vee b_1^1, 2), (x_4 \vee b_2^1, 2), (x_1, 1) \} \cup CNF(\textstyle\sum x_i \leq 1, \infty) \} \cup CNF(b_1^1 + b_2^1 = 1, \infty)\ CNF(b_3^2 + b_4^2 = 1, \infty)\ CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)$$

$$\varphi_{12} = \{ (x_5 \vee b_1^1 \vee b_1^3 \vee b_1^4, 1), (x_4 \vee b_2^1 \vee b_2^3 \vee b_2^4, 1), (x_3 \vee b_3^2 \vee b_3^3 \vee b_3^4, 1), (x_2 \vee b_4^2 \vee b_4^3 \vee b_4^4, 1), (x_5 \vee b_1^1, 2), (x_4 \vee b_2^1, 2), (x_5 \vee b_1^1 \vee b_1^3, 2), (x_4 \vee b_2^1 \vee b_2^3, 2), (x_3 \vee b_3^2 \vee b_3^3, 2), (x_2 \vee b_4^3 \vee b_4^3, 2), (x_1 \vee b_5^4, 1) \} \cup CNF(\textstyle\sum x_i \leq 1, \infty) \cup CNF(b_1^1 + b_2^1 = 1, \infty)\ CNF(b_3^2 + b_4^2 = 1, \infty)\ CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)\ CNF(b_1^4 + b_2^4 + b_3^4 + b_4^4 + b_5^4 = 1, \infty)$$

$\mathcal{I}(\varphi) > 11$

SAT

$\mathcal{I}(\varphi) = 12$

WPM1 algorithm: Stratified Approach

Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$;
$s := 0$; /* Counter of cores */
$w_{max} := decrease(\infty, \varphi)$
while *true* **do**
    $(st, \varphi_c, \mathcal{I}) := SAT(\{C_i \mid (C_i, w_i) \in \varphi[w_i \geq w_{max}]\})$;
    if $st = \text{SAT} \wedge w_{max} = 0$ **then return** $\mathcal{I}(\varphi), \mathcal{I}$;
    if $st = \text{SAT} \wedge w_{max} \neq 0$ **then** $w_{max} := decrease(w_{max}, \varphi)$
    else
        $s := s + 1$;
        $A := \emptyset$; /* Indexes of the core */
        $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$; /* Minimum weight */
        foreach $C_i \in \varphi_c$ **do**
            if $w_i \neq \infty$ **then**
                $b_i^s :=$ new_variable();
                $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b_i^s, w_{min})\}$
                $A := A \cup \{i\}$
            end
        $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A} b_i^s = 1)\}$; /* Cardinal. */
        end
    end
end

Hardening of Soft Clauses in the WPM1 algorithm

Lemma (Lemma 24 in [Ansótegui et al., 2013b])

Let $\varphi_1 = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$ *be a MaxSAT formula with cost zero, let* $\varphi_2 = \{(C'_1, w'_1), \ldots, (C'_r, w'_r)\}$ *be a MaxSAT formula without hard clauses and* $W = \sum_{j=1}^{r} w'_j$. *Let*

$$harden(w) = \begin{cases} w & \text{if } w \leq W \\ \infty & \text{if } w > W \end{cases}$$

and $\varphi'_1 = \{(C_i, harden(w_i)) \mid (C_i, w_i) \in \varphi_1\}$. *Then,* $\mathrm{cost}(\varphi_1 \cup \varphi_2) = \mathrm{cost}(\varphi'_1 \cup \varphi_2)$*, and any optimal assignment for* $\varphi'_1 \cup \varphi_2$ *is an optimal assignment of* $\varphi_1 \cup \varphi_2$.

Hardening in the WPM1 algorithm

$$
\begin{array}{lllllll}
\varphi_{11} = \{ & (x_5 \vee & b_1^1 \vee & & b_1^3, & 3), \\
& (x_4 \vee & b_2^1 \vee & & b_2^3, & 3), \\
& (x_3 \vee & & b_3^2 \vee & b_3^3, & 3), \\
& (x_2 \vee & & b_4^2 \vee & b_4^3, & 3), \\
& (x_5 \vee & b_1^1 & & , & 2), \\
& (x_4 \vee & b_2^1 & & , & 2), \\
& (x_1 & & & , & 1) \} \cup \\
& \multicolumn{6}{l}{CNF(\sum x_i \leq 1, \infty) \} \cup} \\
& \multicolumn{6}{l}{CNF(b_1^1 + b_2^1 = 1, \infty)} \\
& \multicolumn{6}{l}{CNF(b_3^2 + b_4^2 = 1, \infty)} \\
& \multicolumn{6}{l}{CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)}
\end{array}
$$

hardening $w_{max} = 2$
$\mathcal{I}(\varphi) \geq 11$

Hardening in the WPM1 algorithm

$$
\begin{array}{rl}
\varphi_{11} = \{ & (x_5 \lor b_1^1 \lor \ b_1^3, \infty), \\
& (x_4 \lor b_2^1 \lor \ b_2^3, \infty), \\
& (x_3 \lor \ b_3^2 \lor b_3^3, \infty), \\
& (x_2 \lor \ b_4^2 \lor b_4^3, \infty), \\
& (x_5 \lor b_1^1 \qquad\qquad , \infty), \\
& (x_4 \lor b_2^1 \qquad\qquad , \infty), \\
& (x_1 \qquad\qquad\qquad\quad , 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)
\end{array}
$$

hardening $w_{max} = 2$
$\mathcal{I}(\varphi) \geq 11$

Hardening in the WPM1 algorithm

$$
\begin{array}{rl}
\varphi_{11} = \{ & (x_5 \vee b_1^1 \vee \quad b_1^3, \infty), \\
& (x_4 \vee b_2^1 \vee \quad b_2^3, \infty), \\
& (x_3 \vee \quad b_3^2 \vee b_3^3, \infty), \\
& (x_2 \vee \quad b_4^2 \vee b_4^3, \infty), \\
& (x_5 \vee b_1^1 \quad , \quad \infty), \\
& (x_4 \vee b_2^1 \quad , \quad \infty), \\
& (x_1 \quad , \quad 1) \quad \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \quad \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)
\end{array}
$$

$\mathcal{I}(\varphi) > 11$

Hardening in the WPM1 algorithm

$$
\begin{array}{rl}
\varphi_{11} = \{ & (x_5 \vee b_1^1 \vee b_1^3, \infty), \\
& (x_4 \vee b_2^1 \vee b_2^3, \infty), \\
& (x_3 \vee b_3^2 \vee b_3^3, \infty), \\
& (x_2 \vee b_4^2 \vee b_4^3, \infty), \\
& (x_5 \vee b_1^1, \infty), \\
& (x_4 \vee b_2^1, \infty), \\
& (x_1, 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \} \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty)
\end{array}
$$

$$
\begin{array}{rl}
\varphi_{12} = \{ & (x_5 \vee b_1^1 \vee b_1^3, \infty) \\
& (x_4 \vee b_2^1 \vee b_2^3, \infty) \\
& (x_3 \vee b_3^2 \vee b_3^3, \infty) \\
& (x_2 \vee b_4^2 \vee b_4^3, \infty), \\
& (x_5 \vee b_1^1, \infty), \\
& (x_4 \vee b_2^1, \infty), \\
& (x_1 \vee b_5^4, 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(b_1^1 + b_2^1 = 1, \infty) \\
& CNF(b_3^2 + b_4^2 = 1, \infty) \\
& CNF(b_1^3 + b_2^3 + b_3^3 + b_4^3 = 1, \infty) \\
& CNF(b_5^4 = 1, \infty)
\end{array}
$$

$\mathcal{I}(\varphi) > 11$

SAT

$\mathcal{I}(\varphi) = 12$

WPM1 algorithm: Hardening

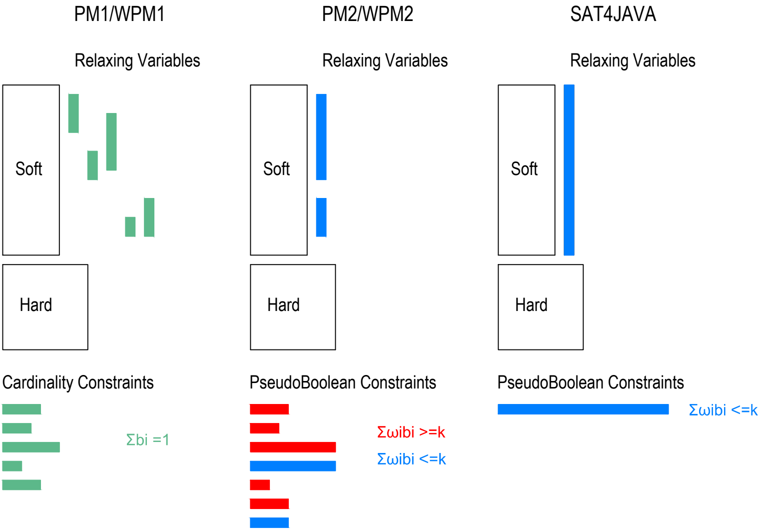
Input: $\varphi = \{(C_1, w_1), \ldots, (C_m, w_m), (C_{m+1}, \infty), \ldots, (C_{m+m'}, \infty)\}$
if $SAT(\{C_i \mid w_i = \infty\}) = (\text{UNSAT}, \_)$ **then return** $(\infty, \emptyset)$;
$s := 0$; /* Counter of cores */
$w_{max} := decrease(\infty, \varphi)$
while *true* **do**
    $(st, \varphi_c, \mathcal{I}) := SAT(\{C_i \mid (C_i, w_i) \in \varphi[w_i \geq w_{max}]\})$;
    if $st = \text{SAT} \wedge w_{max} = 0$ **then return** $\mathcal{I}(\varphi), \mathcal{I}$;
    $W = \sum_{(C_j, w_j) \in \varphi[w_j < w_{max}]} w_j$
    if $st = \text{SAT} \wedge w_{max} \neq 0$ **then** $w_{max} := decrease(w_{max}, \varphi)$
    else
        $s := s + 1$;
        $A := \emptyset$; /* Indexes of the core */
        $w_{min} := \min\{w_i \mid C_i \in \varphi_c \wedge w_i \neq \infty\}$; /* Minimum weight */
        foreach $C_i \in \varphi_c$ **do**
            if $w_i < W$ **then**
                $b_i^s :=$ new_variable();
                $\varphi := \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i, w_i - w_{min}),\ (C_i \vee b_i^s, w_{min})\}$
                $A := A \cup \{i\}$
            end
        $\varphi := \varphi \cup \{(C, \infty) \mid C \in CNF(\sum_{i \in A} b_i^s = 1)\}$; /* Cardinal. */
        end
    end
end

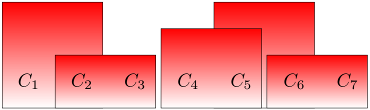
- Original described in [Ansotegui et al., 2010]
- Weighted version of PM2 algorithm [Ansotegui et al., 2009]
- Explores from unsat to sat
- Exploits unsatisfiable cores
- At most one b_i variable per soft clause
- b_i variables added on demand
- Adds PB constraints of the form:

  $\sum w_i \cdot b_i \geq k$ *cores*

  $\sum w_i \cdot b_i \leq k$ *covers*

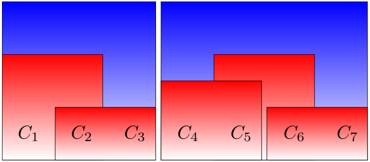
unsat≫sat: the WPM2 algorithm





Covers

Given a set of cores L, we say that A_2 is a *cover* of L,
if it is a nonempty minimal set such that,
for every $A_1 \in L$, if $A_1 \cap A_2 \neq \emptyset$, then $A_1 \subseteq A_2$



Covers

Given a set of cores L, we say that A_2 is a *cover* of L,
if it is a nonempty minimal set such that,
for every $A_1 \in L$, if $A_1 \cap A_2 \neq \emptyset$, then $A_1 \subseteq A_2$

Example: WPM2 on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{ & (x_5 \quad , 5), \\
& (x_4 \quad , 5), \\
& (x_3 \quad , 3), \\
& (x_2 \quad , 3), \\
& (x_1 \quad , 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

$$\varphi_0 = \{\ (x_5\quad, 5), \blacksquare$$
$$(x_4\quad, 5), \blacksquare$$
$$(x_3\quad, 3),$$
$$(x_2\quad, 3),$$
$$(x_1\quad, 1)\ \} \cup$$
$$CNF(\sum x_i \leq 1, \infty)\ \blacksquare$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$;

$$\begin{aligned}\varphi_5 = \{ & (x_5 \vee b_1, 5), \blacksquare \\ & (x_4 \vee b_2, 5), \blacksquare \\ & (x_3, 3), \\ & (x_2, 3), \\ & (x_1, 1)\ \} \cup \\ & CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty)\ \blacksquare \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 \leq 5, \infty)\ \blacksquare\end{aligned}$$

Soft clauses in core are relaxed
$newbound(AL, AM, A) = 5;$
$\mathcal{I}(\varphi) \geq 5;$

Example: WPM2 on PHP^5_1

$$\begin{aligned}\varphi_5 = \{\ &(x_5 \vee b_1 \;\;, 5),\\ &(x_4 \vee b_2 \;\;, 5),\\ &(x_3 \qquad\;\;, 3), \blacksquare\\ &(x_2 \qquad\;\;, 3), \blacksquare\\ &(x_1 \qquad\;\;, 1)\ \} \cup\\ &CNF(\textstyle\sum x_i \leq 1, \infty)\ \blacksquare \cup\\ &CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup\\ &CNF(5 \cdot b_1 + 5 \cdot b_2 \leq 5, \infty)\end{aligned}$$

$\mathcal{I}(\varphi) > 5;$

Example: WPM2 on PHP^5_1

$$
\begin{aligned}
\varphi_8 = \{\ & (x_5 \vee\ b_1\ , 5), \\
& (x_4 \vee\ b_2\ , 5), \\
& (x_3 \vee\ b_3\ , 3), \blacksquare \\
& (x_2 \vee\ b_4\ , 3), \blacksquare \\
& (x_1\ \ \ \ \ \ , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \leq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \leq 3, \infty)\ \blacksquare
\end{aligned}
$$

$newbound(AL, AM, A) = 3;$
$\mathcal{I}(\varphi) \geq 8;$

Example: WPM2 on PHP^5_1

$$\begin{aligned}
\varphi_8 = \{\ & (x_5 \vee\ b_1\ , 5), \\
& (x_4 \vee\ b_2\ , 5), \\
& (x_3 \vee\ b_3\ , 3), \\
& (x_2 \vee\ b_4\ , 3), \\
& (x_1\ \ \ \ \ \ \ , 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \leq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \leq 3, \infty)
\end{aligned}$$

$\mathcal{I}(\varphi) > 8;$

Example: WPM2 on PHP_1^5

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1\ , 5), \\
& (x_4 \vee b_2\ , 5), \\
& (x_3 \vee b_3\ , 3), \\
& (x_2 \vee b_4\ , 3), \\
& (x_1\ \qquad , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)\ \blacksquare
\end{aligned}
$$

$newbound(AL, AM, A) = 11;$
$\mathcal{I}(\varphi) \geq 11;$

One key point in WPM2 is to compute the *$newbound(AL, AM, A)$* which corresponds to the following optimization problem:

$$\text{minimize} \sum_{i \in A} w_i \cdot b_i$$

$$\text{subject to } \{\sum_{i \in A} w_i \cdot b_i > k\} \cup AL$$

where $k = \sum\{k' \mid \sum_{i \in A'} w_i\, b_i \leq k' \in AM \wedge A' \subseteq A\}$ and AM are the at-most constraints corresponding to AL, and A is a cover of the cores of AL.

newbound(AL,AM,A) {

$$\begin{aligned}
&\text{minimize: } 5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \\
&\text{subject to:} \\
&5 \cdot b_1 + 5 \cdot b_2 \geq 5 \\
&3 \cdot b_3 + 3 \cdot b_4 \geq 3 \\
&5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq lb && (4) \\
&lb = 5 + 3 + 1 && (5)
\end{aligned}$$

$$\textit{return } 11, \mathcal{I}_b(b_1, b_2, b_3, b_4) = (0, 1, 1, 1)$$

}

$$\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1\ , 5), \\
& (x_4 \vee b_2\ , 5), \\
& (x_3 \vee b_3\ , 3), \\
& (x_2 \vee b_4\ , 3), \\
& (x_1\ \ \ \ \ \ \ , 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)
\end{aligned}$$

$\mathcal{I}(\varphi) > 11;$

Example: WPM2 on PHP^5_1

$$
\begin{aligned}
\varphi_{12} = \{\ & (x_5 \vee\ b_1\ , 5), \\
& (x_4 \vee\ b_2\ , 5), \\
& (x_3 \vee\ b_3\ , 3), \\
& (x_2 \vee\ b_4\ , 3), \\
& (x_1 \vee\ b_5\ , 1)\ \blacksquare\} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty)\ \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \geq 12, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq 12, \infty)\ \blacksquare
\end{aligned}
$$

$newbound(AL, AM, A) = 12;$
$SAT; \mathcal{I}(\varphi) = 12;$

Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if $st = \text{UNSAT}$ **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
while *true* **do**
  if $\sum_{lb \in am \in AM} lb = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi, AL, AM)$
  if $st = \text{SAT}$ **then**
    $\mathcal{I}_u = \mathcal{I}$
  end
  else
    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c)\}$ \\ New core
    $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$ \\ New cover
    $(lb, \mathcal{I}_b) := newbound(AL, AM, A)$
    $AL := AL \cup \{\sum_{i \in A} w_i\, b_i \geq lb\}$
    $AM := \{am \in AM \mid indexs(am) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \leq lb\}$
  end
end

WPM2: stratification

Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if $st =$ UNSAT **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
$w_{max} := decrease(\infty, \varphi)$
while *true* **do**
  if $\sum_{lb \in am \in AM} lb = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi[w_i \geq w_{max}], AL, AM)$
  if $st =$ SAT **then**
    $\mathcal{I}_u = \mathcal{I}$
    $w_{max} := decrease(w_{max}, \varphi)$
  end
  else
    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c)\}$ \\ New core
    $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$ \\ New cover
    $(lb, \mathcal{I}_b) := newbound(AL, AM, A)$
    $AL := AL \cup \{\sum_{i \in A} w_i\, b_i \geq lb\}$
    $AM := \{am \in AM \mid indexs(am) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \leq lb\}$
  end
end

WPM2: stratification and hardening

Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if $st =$ UNSAT **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
$w_{max} := decrease(\infty, \varphi)$
while *true* **do**
  if $\sum_{lb \in am \in AM} lb = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi[w_i \geq w_{max}], AL, AM)$
  if $st =$ SAT **then**
    $\mathcal{I}_u = \mathcal{I}$
    $\varphi_h := harden(\varphi, AM, \sum_{(C_i, w_i) \in \varphi[w_i < w_{max}]} w_i)$
    $w_{max} := decrease(w_{max}, \varphi)$
  end
  else
    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c \backslash \varphi_h)\}$ \\ New core
    $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$ \\ New cover
    $(lb, \mathcal{I}_b) := newbound(AL, AM, A)$
    $AL := AL \cup \{\sum_{i \in A} w_i\, b_i \geq lb\}$
    $AM := \{am \in AM \mid indexs(am) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \leq lb\}$
  end
end

Hardening in the WPM2 algorithm

$$\varphi_{11} = \{ (x_5 \vee b_1, 5), (x_4 \vee b_2, 5), (x_3 \vee b_3, 3), (x_2 \vee b_4, 3), (x_1, 1) \} \cup$$
$$CNF(\sum x_i \leq 1, \infty) \cup$$
$$CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup$$
$$CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup$$
$$CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup$$
$$CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)$$

hardening $w_{max} = 3$;
newbound$(\emptyset, AM, A) = 13$; $13 - 11 > 1$;
$\mathcal{I}(\varphi) \geq 11$;

Hardening in the WPM2 algorithm

$$\begin{aligned}\varphi_{11} = \{\ & (x_5 \vee\ b_1\ ,\ \infty), \\ & (x_4 \vee\ b_2\ ,\ \infty), \\ & (x_3 \vee\ b_3\ ,\ \infty), \\ & (x_2 \vee\ b_4\ ,\ \infty), \\ & (x_1 \qquad\ ,\ 1)\ \} \cup \\ & CNF(\sum x_i \leq 1, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\ & CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup \\ & CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)\end{aligned}$$

$hardening\ w_{max} = 3;$
$newbound(\emptyset, AM, A) = 13;\ \ 13 - 11 > 1;$
$\mathcal{I}(\varphi) \geq 11;$

Hardening in the WPM2 algorithm

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1\ , \infty), \blacksquare \\
& (x_4 \vee b_2\ , \infty), \blacksquare \\
& (x_3 \vee b_3\ , \infty), \blacksquare \\
& (x_2 \vee b_4\ , \infty), \blacksquare \\
& (x_1\ \quad , 1) \quad \blacksquare \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)\ \blacksquare
\end{aligned}
$$

$\mathcal{I}(\varphi) > 11;$

Hardening in the WPM2 algorithm

$$
\begin{aligned}
\varphi_{12} = \{ & (x_5 \vee b_1, \infty), \\
& (x_4 \vee b_2, \infty), \\
& (x_3 \vee b_3, \infty), \\
& (x_2 \vee b_4, \infty), \\
& (x_1 \vee b_5, 1) \; \blacksquare \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup \\
& CNF(1 \cdot b_5 \geq 1, \infty) \; \blacksquare \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty) \\
& CNF(1 \cdot b_5 \leq 1, \infty) \; \blacksquare
\end{aligned}
$$

$SAT; \mathcal{I}(\varphi) = 12;$

Lemma

Let $\varphi_1 \cup \varphi_2$ *be a MaxSAT formula and* k_1 *and* k_2 *values such that:* $\mathrm{cost}(\varphi_1 \cup \varphi_2) = k_1 + k_2$ *and any assignment* I *satisfies* $I(\varphi_1) \geq k_1$ *and* $I(\varphi_2) \geq k_2$.

Let k' *be the smallest possible optimal of* φ_2 *such that* $k' > k_2$. *Let* φ_3 *be a set of soft clauses with* $W = \sum\{w_i \mid (C_i, w_i) \in \varphi_3\}$.

Then, if $W < k' - k_2$, *then any optimal assignment* I' *of* $\varphi_1 \cup \varphi_2 \cup \varphi_3$ *assigns* $I'(\varphi_2) = k_2$

Comparison with check-optimum solver

	wpm2	*wpm1*	*checkOp*	*sat4j*	*shinms*	*qms*
Industrial PMS	428	330	291	289	353	418
Industrial WPMS	207	204	22	18	52	-
Industrial TOTAL	635	534	313	307	405	-
Crafted PMS	268	207	222	221	271	291
Crafted WPMS	271	318	199	196	264	-
Crafted TOTAL	539	525	421	417	535	-

Table: Comparison with check-optimum: solved instances.

Comparison with check-optimum solver

	wpm2	*wpm1*	*checkOp*	*sat4j*	*shinms*	*qms*
Industrial PMS	428	330	291	289	353	418
	77.6%	-	100.0%	100.0%		
Industrial WPMS	207	204	22	18	52	-
	9.0%	-	100.0%	100.0%		
Industrial TOTAL	635	534	313	307	405	-
	64.7%	-	100.0%	100.0%		
Crafted PMS	268	207	222	221	271	291
	91.1%	-	100.0%	100.0%		
Crafted WPMS	271	318	199	196	264	-
	86.1%	-	100.0%	100.0%		
Crafted TOTAL	539	525	421	417	535	-
	88.6%	-	100.0%	100.0%		

Table: Comparison with check-optimum: % of relaxed clauses.

Cover optimization in the WPM2 algorithm

As we mentioned, one key point in WPM2 is how to compute the new lower bound for a cover A

Cover optimization [Ansótegui et al., 2013a]: solve the maxsat problem represented by soft clauses in cover A and hard clauses

WPM2 is parametric on the cover optimization technique

In practice, a model-guided approach is used (sat4java like)

More on Thursday ...

Cover Optimization in the WPM2 algorithm

$$\begin{aligned}\varphi_0 = \{ & (x_5 \quad, 5), \blacksquare \\ & (x_4 \quad, 5), \blacksquare \\ & (x_3 \quad, 3), \blacksquare \\ & (x_2 \quad, 3), \blacksquare \\ & (x_1 \quad, 1) \} \cup \\ & CNF(\sum x_i \leq 1, \infty) \blacksquare\end{aligned}$$

SAT(φ_0) returns a core

$\mathcal{I}(\varphi) > 0$;

Cover Optimization in the WPM2 algorithm

$$\begin{aligned} \varphi_3 = \{\ & (x_5 \vee b_1\ , 5), \\ & (x_4 \vee b_2\ , 5), \\ & (x_3 \vee b_3\ , 3), \\ & (x_2 \vee b_4\ , 3), \\ & (x_1\ , 1)\ \} \cup \\ & CNF(\sum x_i \leq 1, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 3, \infty) \end{aligned}$$

$newbound(AL, AM, A) = 3;$
$\mathcal{I}(\varphi) \geq 3;$

Cover Optimization in the WPM2 algorithm

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1 \ , 5), \blacksquare \\
& (x_4 \vee b_2 \ , 5), \blacksquare \\
& (x_3 \vee b_3 \ , 3), \blacksquare \\
& (x_2 \vee b_4 \ , 3), \blacksquare \\
& \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty) \ \blacksquare
\end{aligned}
$$

$newbound(AL \cup \varphi[i \in 1, \ldots, 4], AM, A) = 11;$
$\mathcal{I}(\varphi) \geq 11;$

Cover Optimization in the WPM2 algorithm

$$\begin{aligned}\varphi_{11} = \{\ & (x_5 \vee b_1\ , 5), \\ & (x_4 \vee b_2\ , 5), \\ & (x_3 \vee b_3\ , 3), \\ & (x_2 \vee b_4\ , 3), \\ & (x_1\ \quad\quad , 1)\ \ \} \cup \\ & CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\ & CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\ & CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty) \cup \\ & CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 11, \infty)\end{aligned}$$

$\mathcal{I}(\varphi) > 11;$

$$
\begin{aligned}
\varphi_{12} = \{\ & (x_5 \vee\ b_1\ , 5), \\
& (x_4 \vee\ b_2\ , 5), \\
& (x_3 \vee\ b_3\ , 3), \\
& (x_2 \vee\ b_4\ , 3), \\
& (x_1 \vee\ b_5\ , 1)\ \blacksquare \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \geq 5, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \geq 3, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \geq 11, \infty)\ \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \geq 12, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 3 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq 12, \infty)\ \blacksquare
\end{aligned}
$$

SAT;
$\mathcal{I}(\varphi) = 12$;

WPM2: stratification, hardening and cover optimization

Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if st = UNSAT **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
$w_{max} := decrease(\infty, \varphi)$
while *true* **do**

  if $\sum_{lb \in am \in AM} lb = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi[w_i \geq w_{max}], AL, AM)$
  if st = SAT **then**

    $\mathcal{I}_u = \mathcal{I}$
    $\varphi_h := harden(\varphi, AM, \sum_{(C_i, w_i) \in \varphi[w_i < w_{max}]} w_i)$
    $w_{max} := decrease(w_{max}, \varphi)$

  end
  else

    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c \backslash \varphi_h)\}$ \\ New core
    $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$ \\ New cover
    $(lb, \mathcal{I}_b) := newbound(AL \cup \varphi[i \in A \vee w_i = \infty], AM, A)$
    $AL := AL \cup \{\sum_{i \in A} w_i\, b_i \geq lb\}$
    $AM := \{am \in AM \mid indexs(am) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \leq lb\}$

  end

end

Experimental results

solvers	pms	wpms	Ind.	pms	wpms	Cra.	Total
wpm2	374 69.6%	130 50.9%	504 66.1%	246 64.9%	46 19.5%	292 42.2%	796 54.1%
$wpm2_s$	376 70.9%	133 51.9%	509 67.3%	247 65.1%	114 31.9%	361 48.5%	870 57.9%
$wpm2_{sh}$	376 70.9%	198 73.6%	574 71.4%	247 65.1%	115 32.5%	362 48.8%	936 60.1%

Table: WPM2: stratification (s) and hardening (h)

Experimental results

solvers	pms	wpms	Ind.	pms	wpms	Cra.	Total
wpm2	374 69.6%	130 50.9%	504 66.1%	246 64.9%	46 19.5%	292 42.2%	796 54.1%
$wpm2_s$	376 70.9%	133 51.9%	509 67.3%	247 65.1%	114 31.9%	361 48.5%	870 57.9%
$wpm2_{sh}$	376 70.9%	198 73.6%	574 71.4%	247 65.1%	115 32.5%	362 48.8%	936 60.1%
$wpm2_{shla}$	394 76.0%	201 75.5%	595 75.9%	254 66.5%	221 52.6%	475 59.5%	1070 67.7%
$wpm2_{shba}$	415 81.4%	203 75.3%	618 80.2%	260 67.4%	281 63.4%	541 65.4%	1159 72.8%
$wpm2_{shua}$	428 83.6%	207 77.5%	635 82.5%	268 68.5%	271 62.0%	539 65.3%	1174 73.9%

Table: WPM2: cover optimization for all covers (a)

(l)	cover optimization from lower bound
(b)	cover optimization with binary search
(u)	cover optimization from upper bound

Experimental results

solvers	pms	wpms	Ind.	pms	wpms	Cra.	Total
wpm2	374 69.6%	130 50.9%	504 66.1%	246 64.9%	46 19.5%	292 42.2%	796 54.1%
$wpm2_s$	376 70.9%	133 51.9%	509 67.3%	247 65.1%	114 31.9%	361 48.5%	870 57.9%
$wpm2_{sh}$	376 70.9%	198 73.6%	574 71.4%	247 65.1%	115 32.5%	362 48.8%	936 60.1%
$wpm2_{shlc}$	387 72.5%	201 75.5%	588 73.0%	251 65.9%	219 52.2%	470 59.1%	1058 66.1%
$wpm2_{shla}$	394 76.0%	201 75.5%	595 75.9%	254 66.5%	221 52.6%	475 59.5%	1070 67.7%
$wpm2_{shbc}$	404 77.5%	205 76.9%	609 77.4%	261 67.6%	280 62.9%	541 65.2%	1150 71.3%
$wpm2_{shba}$	415 81.4%	203 75.3%	618 80.2%	260 67.4%	281 63.4%	541 65.4%	1159 72.8%
$wpm2_{shuc}$	425 81.7%	206 76.3%	631 80.7%	266 68.4%	273 62.0%	539 65.2%	1170 72.9%
$wpm2_{shua}$	428 83.6%	207 77.5%	635 82.5%	268 68.5%	271 62.0%	539 65.3%	1174 73.9%

Table: WPM2: cover optimization for repeated covers (c)

(l)	cover optimization from lower bound
(b)	cover optimization with binary search
(u)	cover optimization from upper bound

- Originally described in [Davies and Bacchus, 2011]
- Explores from unsat to sat
- Exploits unsatisfiable cores (core-guided)
- Hybrid approach of SAT and MIP
- No PB constraints to the SAT solver
- The arithmetic is performed by a MIP solver
- *More* SAT calls, but *simpler* ...

Example: MaxHS on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{ & (x_5 \quad , 5), \\
& (x_4 \quad , 5), \\
& (x_3 \quad , 3), \\
& (x_2 \quad , 3), \\
& (x_1 \quad , 1), \\
\} & \cup \; CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

Example: MaxHS on PHP_1^5

$$\varphi_0 = \{ (x_5 \quad, 5), \blacksquare$$
$$(x_4 \quad, 5), \blacksquare$$
$$(x_3 \quad, 3),$$
$$(x_2 \quad, 3),$$
$$(x_1 \quad, 1),$$
$$\} \cup \; CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 5 \cdot b_4$
subject to:
$b_5 + b_4 \geq 1$

$\mathcal{I}(b_5, b_4) = (1, 0);$
erase $(x_5, 5);\ \mathcal{I}(\varphi) \geq 5;$

Example: MaxHS on PHP_1^5

$$
\begin{aligned}
\varphi_5 = \{\ & (x_4 \quad , 5), \\
& (x_3 \quad , 3), \\
& (x_2 \quad , 3), \\
& (x_1 \quad , 1), \\
\} \cup\ & CNF(\textstyle\sum x_i \leq 1, \infty)
\end{aligned}
$$

minimize: $5 \cdot b_5 + 5 \cdot b_4 + 3 \cdot b_3$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$

$\mathcal{I}(b_5, b_4, b_3) = (0, 1, 0)$;
erase $(x_4, 5)$; $\mathcal{I}(\varphi) \geq 5$;

$$\varphi_5 = \{\ (x_5\ \ , 5), \blacksquare$$
$$(x_3\ \ , 3), \blacksquare$$
$$(x_2\ \ , 3),$$
$$(x_1\ \ , 1),$$
$$\} \cup CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 5 \cdot b_4 + 3 \cdot b_3$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$

$\mathcal{I}(b_5, b_4, b_3) = (1, 0, 1)$;
erase $(x_5, 5), (x_3, 3)$; $\mathcal{I}(\varphi) \geq 8$;

$$\varphi_8 = \{ (x_4 \quad , 5), \quad (x_2 \quad , 3), \quad (x_1 \quad , 1), \quad \} \cup CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 5 \cdot b_4 + 3 \cdot b_3 + 3 \cdot b_2$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$
$b_4 + b_2 \geq 1$

$\mathcal{I}(b_5, b_4, b_3, b_2) = (0, 1, 1, 0)$;
erase $(x_4, 5), (x_3, 3)$; $\mathcal{I}(\varphi) \geq 8$;

Example: MaxHS on PHP^5_1

$$\begin{aligned}\varphi_8 = \{\ & (x_5 \quad , 5), \blacksquare \\ & (x_2 \quad , 3), \blacksquare \\ & (x_1 \quad , 1), \\ \}\ \cup\ & CNF(\textstyle\sum x_i \leq 1, \infty)\end{aligned}$$

minimize: $5 \cdot b_5 + 5 \cdot b_4 + 3 \cdot b_3 + 3 \cdot b_2$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$
$b_4 + b_2 \geq 1$
$b_5 + b_2 \geq 1$

$\mathcal{I}(b_5, b_4, b_3, b_2) = (1, 1, 0, 0)$;
erase $(x_5, 5), (x_4, 5)$; $\mathcal{I}(\varphi) \geq 10$;

$$\varphi_{10} = \{ (x_3 \quad , 3), \quad (x_2 \quad , 3), \quad (x_1 \quad , 1), \} \cup CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 5 \cdot b_4 + 3 \cdot b_3 + 3 \cdot b_2 + 1 \cdot b_1$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$
$b_4 + b_2 \geq 1$
$b_5 + b_2 \geq 1$
$b_3 + b_2 \geq 1$

$\mathcal{I}(b_5, b_4, b_3, b_2, b_1) = (1, 0, 1, 1, 0)$;
erase $(x_5, 5), (x_3, 3), (x_2, 3)$; $\mathcal{I}(\varphi) \geq 11$;

Example: MaxHS on PHP^5_1

$$\varphi_{11} = \{\ (x_4 \quad, 5), \blacksquare$$
$$(x_1 \quad, 1), \blacksquare$$
$$\} \cup\ CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 4 \cdot b_4 + 3 \cdot b_3 + 2 \cdot b_2 + 1 \cdot b_1$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$
$b_4 + b_2 \geq 1$
$b_5 + b_2 \geq 1$
$b_3 + b_2 \geq 1$
$b_4 + b_1 \geq 1$

$\mathcal{I}(b_5, b_4, b_3, b_2, b_1) = (0, 1, 1, 1, 0)$;
erase $(x_4, 5), (x_3, 3), (x_2, 3)$; $\mathcal{I}(\varphi) \geq 11$;

$$\varphi_{11} = \{\ (x_5\ \ ,5), \blacksquare$$
$$(x_1\ \ ,1), \blacksquare$$
$$\} \cup\ CNF(\sum x_i \leq 1, \infty)$$

minimize: $5 \cdot b_5 + 4 \cdot b_4 + 3 \cdot b_3 + 2 \cdot b_2 + 1 \cdot b_1$
subject to:
$b_5 + b_4 \geq 1$
$b_4 + b_3 \geq 1$
$b_5 + b_3 \geq 1$
$b_4 + b_2 \geq 1$
$b_5 + b_2 \geq 1$
$b_3 + b_2 \geq 1$
$b_4 + b_1 \geq 1$
$b_5 + b_1 \geq 1$

$\mathcal{I}(b_5, b_4, b_3, b_2, b_1) = (0, 1, 1, 1, 1)$
erase $(x_4, 5), (x_3, 3), (x_2, 3), (x_1, 1)$; $\mathcal{I}(\varphi) \geq 12$;

Example: MaxHS on PHP_1^5

$$\varphi_{12} = \{\ (x_5 \quad , 5), \\ \} \cup\ CNF(\sum x_i \leq 1, \infty)$$

SAT; $\mathcal{I}(\varphi) = 12$;
sequence of lower bounds $= 0, 5, 5, 8, 8, 10, 11, 11, 12$

MAXHS algorithm

Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if $st = \text{UNSAT}$ **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
while *true* **do**
  if $\sum_{k \in am \in AM} k = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi, \emptyset, AM)$
  if $st = \text{SAT}$ **then**
    $\mathcal{I}_u = \mathcal{I}$
  end
  else
    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c)\}$ \\ New core
    $(lb, \mathcal{I}_b) := newbound(AL \cup \{\sum_{i \in A} b_i \geq 1\}, \emptyset, indexs(AL))$
    $AL := AL \cup \{\sum_{i \in A} b_i \geq 1\}$
    $AM := \bigcup_{i \in indexs(AL) \wedge \mathcal{I}_b(b_i) = 1} w_i, b_i \leq w_i$
  end
end

Obsv: ILP solves efficiently MaxSAT

Idea: move some hard clauses ILP

Jessica Davies, Fahiem Bacchus:

Exploiting the Power of mip Solvers in maxsat. SAT 2013: 166-181

MaxHS: new improvements

Obsv: ILP solves efficiently MaxSAT

Idea: move some hard clauses ILP

Jessica Davies, Fahiem Bacchus:
Exploiting the Power of mip Solvers in maxsat. SAT 2013: 166-181

Obsv: calls to ILP become harder with hard clauses

Idea: postpone calls to ILP

Jessica Davies and Fahiem Bacchus:
Postponing Optimization to Speed Up MAXSAT Solving. CP 2013
(Solver at MaxSAT Evaluation 2013)

More on Thursday ...

- Originally described in BINCD [Heras et al., 2011]
- New version BINCD2 [Morgado et al., 2012]
- Binary Core Driven search
- Exploits unsatisfiable cores and satisfying assignments
- Keeps a *lb* and *ub* for each cover *c*
- Next k is $\sum_{c \in covers} \frac{c.lb + c.ub}{2}$
- At most one b_i variable per soft clause
- b_i variables added on demand
- Adds PB constraints of the form:

$$\sum w_i \cdot b_i \geq k$$
$$\sum w_i \cdot b_i \leq k$$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_0 = \{ & (x_5 \quad , 5), \\
& (x_4 \quad , 5), \\
& (x_3 \quad , 3), \\
& (x_2 \quad , 3), \\
& (x_1 \quad , 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty)
\end{aligned}
$$

Example: BINCD2 on PHP_1^5

$$
\begin{aligned}
\varphi_0 = \{ & (x_5 \quad, 5), \\
& (x_4 \quad, 5), \\
& (x_3 \quad, 3), \\
& (x_2 \quad, 3), \\
& (x_1 \quad, 1)\ \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty)
\end{aligned}
$$

SAT(φ_0) returns a core

$al_1.lb = 5; am_1.k = 7; am_1.ub = 10$
$\mathcal{I}(\varphi) \geq 5;$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_7 = \{ & (x_5 \vee b_1 \;, 5), \blacksquare \\
& (x_4 \vee b_2 \;, 5), \blacksquare \\
& (x_3 \;, 3), \\
& (x_2 \;, 3), \\
& (x_1 \;, 1) \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 4 \cdot b_2 \leq 7, \infty) \;\blacksquare
\end{aligned}
$$

Soft clauses in core are relaxed

$\mathcal{I}(\varphi) \geq 5$;

Example: BINCD2 on PHP^5_1

$$\varphi_7 = \{ (x_5 \vee b_1 \ , 5),$$
$$(x_4 \vee b_2 \ , 5),$$
$$(x_3 \ , 3), \blacksquare$$
$$(x_2 \ , 3), \blacksquare$$
$$(x_1 \ , 1) \} \cup$$
$$CNF(\sum x_i \leq 1, \infty) \ \blacksquare \cup$$
$$CNF(5 \cdot b_1 + 4 \cdot b_2 \leq 7, \infty)$$

SAT(φ_1) returns a core

$al_1.lb = 5; am_1.k = 7; am_1.ub = 10$
$al_2.lb = 3; am_2.k = 4; am_2.ub = 6$
$\mathcal{I}(\varphi) \geq 8;$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_{11} = \{ \ & (x_5 \vee \ b_1 \ , 5), \\
& (x_4 \vee \ b_2 \ , 5), \\
& (x_3 \vee \ b_3 \ , 3), \ \blacksquare \\
& (x_2 \vee \ b_4 \ , 3), \ \blacksquare \\
& (x_1 \qquad \ , 1) \ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 \leq 7, \infty) \cup \\
& CNF(3 \cdot b_3 + 3 \cdot b_4 \leq 4, \infty) \ \ \blacksquare
\end{aligned}
$$

Soft clauses in core are relaxed

$\mathcal{I}(\varphi) \geq 8;$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee\ b_1\ , 5),\ \blacksquare \\
& (x_4 \vee\ b_2\ , 5),\ \blacksquare \\
& (x_3 \vee\ b_3\ , 3),\ \blacksquare \\
& (x_2 \vee\ b_4\ , 3),\ \blacksquare \\
& (x_1 \qquad\ , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty)\ \blacksquare \cup \\
& CNF(5 \cdot b_1 + 4 \cdot b_2 \leq 7, \infty)\ \blacksquare \cup \\
& CNF(3 \cdot b_3 + 2 \cdot b_4 \leq 4, \infty)\ \blacksquare
\end{aligned}
$$

A new cover is computed

$al_1.lb = 5; am_1.k = 7; am_1.ub = 10$
$al_2.lb = 3; am_2.k = 4; am_2.ub = 6$
$al_3.lb = 8 + 1 + 1; am_3.k = 13; am_3.ub = 16$
$\mathcal{I}(\varphi) \geq 10;$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_{13} = \{\ & (x_5 \vee b_1\ , 5), \\
& (x_4 \vee b_2\ , 5), \\
& (x_3 \vee b_3\ , 3), \\
& (x_2 \vee b_4\ , 3), \\
& (x_1 \qquad , 1)\ \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 13, \infty)
\end{aligned}
$$

Overlaped AMs are replaced for new cover AM

$\mathcal{I}(\varphi) \geq 10;$

$$\begin{aligned}\varphi_{13} = \{\ &(x_5 \vee\ b_1\ , 5),\ \blacksquare\\ &(x_4 \vee\ b_2\ , 5),\ \blacksquare\\ &(x_3 \vee\ b_3\ , 3),\ \blacksquare\\ &(x_2 \vee\ b_4\ , 3),\ \blacksquare\\ &(x_1\ \quad\quad , 1)\ \blacksquare\} \cup\\ &CNF(\sum x_i \leq 1, \infty)\ \blacksquare \cup\\ &CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 \leq 13, \infty)\ \blacksquare\end{aligned}$$

A new cover is computed

$al_3.lb = 10; am_3.k = 13; am_3.ub = 16$
$al_4.lb = 10 + 1; am_4.k = 14; am_4.ub = 17$
$\mathcal{I}(\varphi) \geq 11;$

$$
\begin{aligned}
\varphi_{14} = \{\ & (x_5 \vee b_1\ , 5),\\
& (x_4 \vee b_2\ , 5),\\
& (x_3 \vee b_3\ , 3),\\
& (x_2 \vee b_4\ , 3),\\
& (x_1 \vee b_5\ , 1)\ \blacksquare \} \cup\\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup\\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq 14, \infty)\ \blacksquare
\end{aligned}
$$

Overlaped AM is replaced for new cover AM

SAT
$\mathcal{I}(\varphi) \leq 14$;
$\mathcal{I}(\varphi) \geq 11$;

$$
\begin{aligned}
\varphi_{12} = \{\ & (x_5 \vee b_1\ , 5), \\
& (x_4 \vee b_2\ , 5), \\
& (x_3 \vee b_3\ , 3), \\
& (x_2 \vee b_4\ , 3), \\
& (x_1 \vee b_5\ , 1)\ \blacksquare \} \cup \\
& CNF(\textstyle\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq 12, \infty)\ \blacksquare
\end{aligned}
$$

SAT
$\mathcal{I}(\varphi) \leq 12;$
$\mathcal{I}(\varphi) \geq 11;$

Example: BINCD2 on PHP^5_1

$$
\begin{aligned}
\varphi_{11} = \{\ & (x_5 \vee b_1\ , 5), \\
& (x_4 \vee b_2\ , 5), \\
& (x_3 \vee b_3\ , 3), \\
& (x_2 \vee b_4\ , 3), \\
& (x_1 \vee b_5\ , 1)\ \blacksquare \} \cup \\
& CNF(\sum x_i \leq 1, \infty) \cup \\
& CNF(5 \cdot b_1 + 5 \cdot b_2 + 3 \cdot b_3 + 3 \cdot b_4 + 1 \cdot b_5 \leq 11, \infty)\ \blacksquare
\end{aligned}
$$

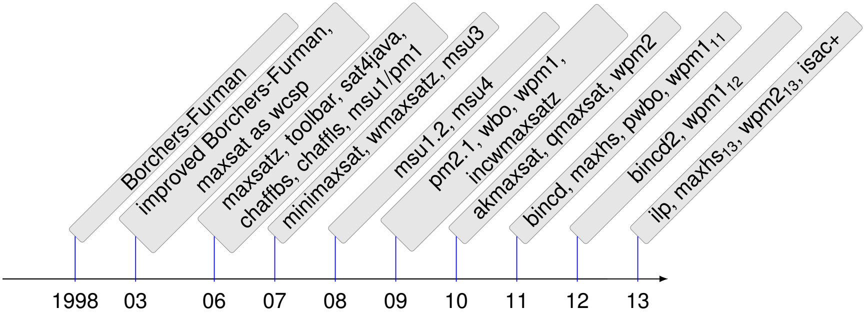
UNSAT
$\mathcal{I}(\varphi) > 11;$
$\mathcal{I}(\varphi) = 12;$

BINCD2 algorithm

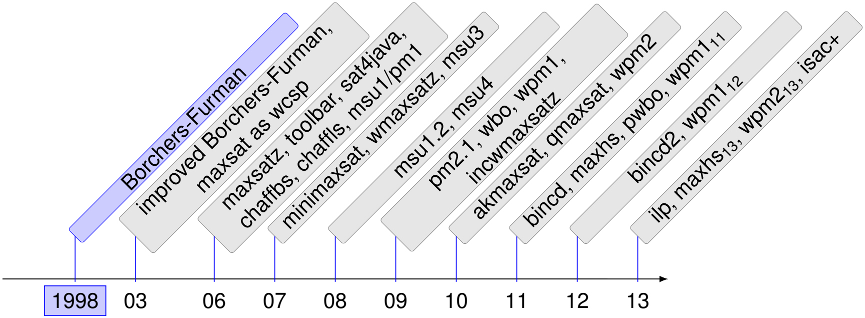
Input: $\varphi = \{(C_1, w_1), \ldots, (C_s, w_s), (C_{s+1}, \infty), \ldots, (C_{s+h}, \infty)\}$
$B := \{b_1, \ldots, b_s\}$
$(st, \varphi_c, \mathcal{I}_u) := sat(\varphi[w_i = \infty], \emptyset, \emptyset)$
if $st = \text{UNSAT}$ **then return** $(\infty, \emptyset)$
$AL := \emptyset$
$AM := \emptyset$
while *true* **do**
  if $\sum_{lb \in al \in AL} lb = \mathcal{I}_u(\varphi)$ **then return** $(\mathcal{I}_u(\varphi), \mathcal{I}_u)$
  $(st, \varphi_c, \mathcal{I}) := sat(\varphi, \emptyset, AM)$
  if $st = \text{SAT}$ **then**
    $\mathcal{I}_u = \mathcal{I}$
    $AM := \{(\sum_{i \in A} w_i\, b_i \leq lb + \beta(\mathcal{I}(\varphi[i \in A] - lb) \mid (\sum_{i \in A} w_i\, b_i \geq lb) \in AL$
  end
  else
    $A := \{i \mid (C_i \vee b_i) \in (\varphi_c)\}$ \\ New core
    $A := \bigcup_{\substack{A' \in cores(AL) \\ A' \cap A \neq \emptyset}} A'$ \\ New cover
    $lb := newbound_{\Delta}(AL, AM, A)$
    $k := lb + \beta(\mathcal{I}_u(\varphi[i \in A] - lb)$
    $AL := \{al \in AL \mid indexs(al) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \geq lb\}$
    $AM := \{am \in AM \mid indexs(am) \cap A = \emptyset\} \cup \{\sum_{i \in A} w_i\, b_i \leq k\}$
  end
end

- Described in [Martins et al., 2012]
- Two threads are executed
- First thread searches from unsat to sat, as WBO
- Second thread searches from sat to unsat, as SAT4J
- Learned clauses are shared between both threads

Exact MaxSAT Solvers Timeline



Exact MaxSAT Solvers Timeline



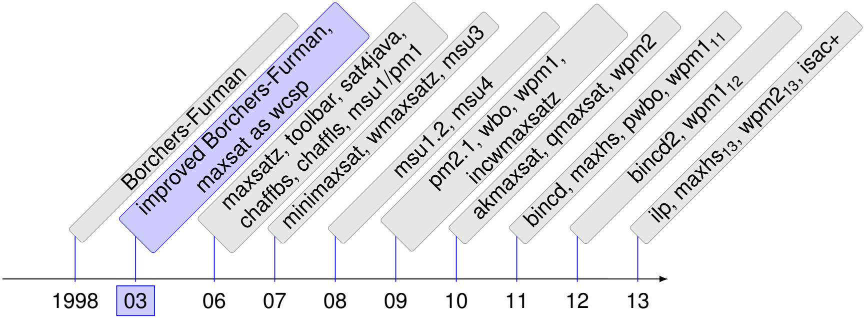
1998: **Borchers-Furman** [Borchers and Furman, 1998]

Contribution: B&B MaxSat solver based on DPLL. GSAT for initial UB

MaxSAT Evaluations: -

Brian Borchers, Judith Furman: **A Two-Phase Exact Algorithm for MAX-SAT and Weighted MAX-SAT Problems.** J. Comb. Optim. 2(4): 299-306 (1998)

Exact MaxSAT Solvers Timeline



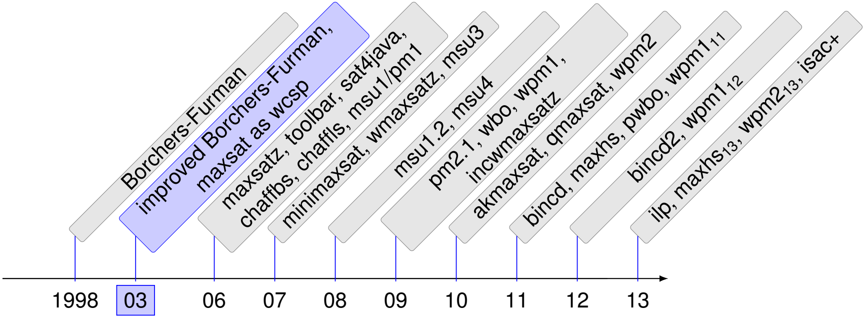
2003: **improved Borchers-Furman** [Alsinet et al., 2003]

Contribution: Improved Borchers-Furman with underestimation

MaxSAT Evaluations: -

Teresa Alsinet and Felip Manyà and Jordi Planes: **Improved Branch and Bound Algorithms for Max-SAT** Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing, 2003

Exact MaxSAT Solvers Timeline



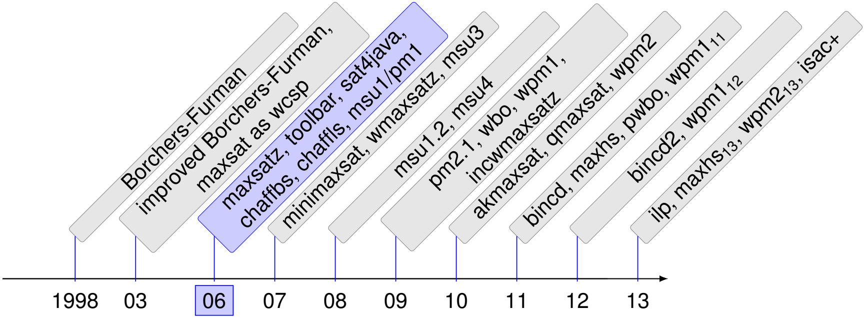
2003: **maxsat as wcsp** [de Givry et al., 2003]

Contribution: WCSP techniques for solving MaxSAT

MaxSAT Evaluations: -

Simon de Givry, Javier Larrosa, Pedro Meseguer, Thomas Schiex: **Solving Max-SAT as Weighted CSP.** CP 2003: 363-376

Exact MaxSAT Solvers Timeline

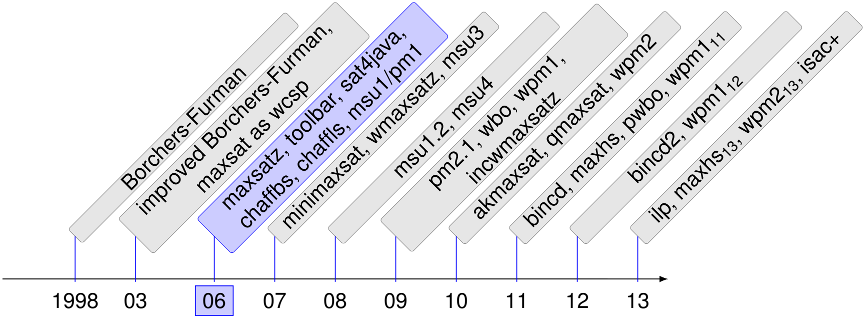


2006: **maxsatz** [Li et al., 2007]

Contribution: UP-based underestimation and incomplete resolution rules

MaxSAT Evaluations: from 2006

Chu Min Li, Felip Manya, Jordi Planes: **New Inference Rules for Max-SAT.** J. Artif. Intell. Res. (JAIR) 30: 321-359 (2007)



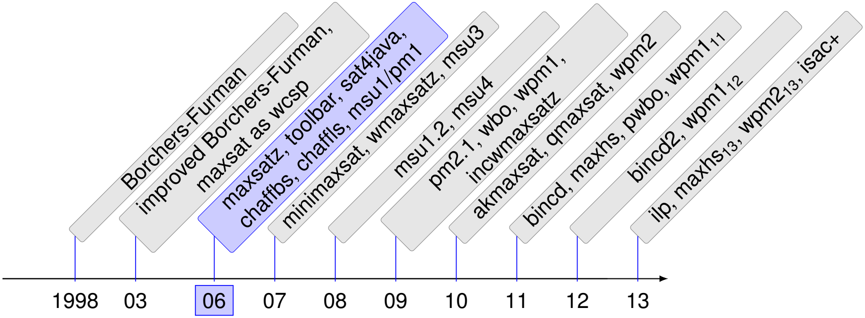
2006: **toolbar** [Sanchez et al., 2008]

Contribution: local consistency techniques for WCSP

MaxSAT Evaluations: 2006, 2007, 2008

M. Sanchez, S. Bouveret, S. De Givry, F. Heras, P. Jégou, J. Larrosa, S. Ndiaye, E. Rollon, T. Schiex, C. Terrioux, G. Verfaillie, M. Zytnicki: **Max-CSP competition 2008: toulbar2 solver description.**

Exact MaxSAT Solvers Timeline



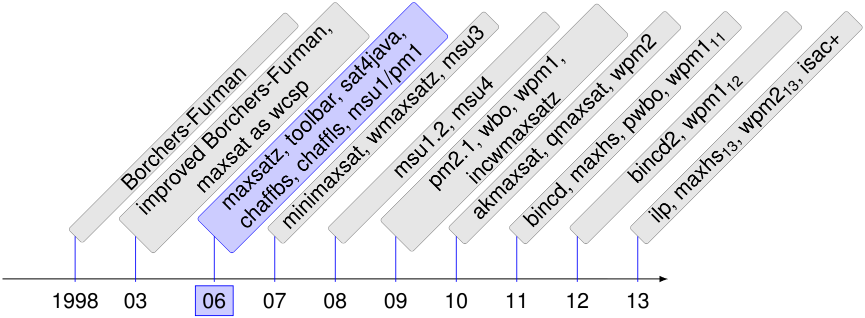
2006: **sat4java** [Berre, 2006]

Contribution: model-guided approach for MaxSAT (as minisat+ for PB)

MaxSAT Evaluations: from 2006

Daniel Le Berre: **SAT4J, a satisfiability library for java**, 2006

Exact MaxSAT Solvers Timeline



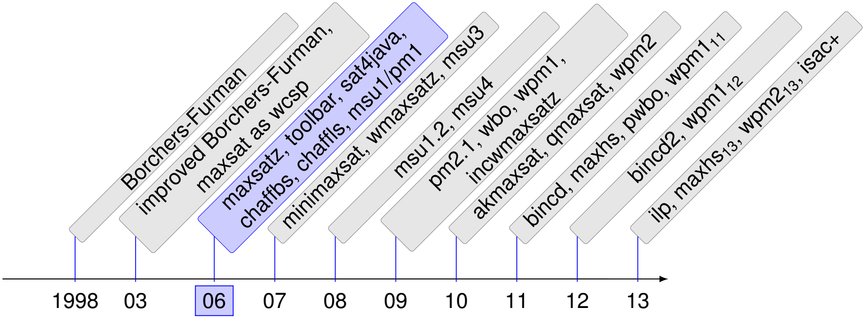
2006: **chaffbs, chaffls** [Fu and Malik, 2006]

Contribution: binary SAT-based approach (chaffbs), linear up SAT-based (chaffls)

MaxSAT Evaluations: 2006

Zhaohui Fu, Sharad Malik: **On Solving the Partial MAX-SAT Problem.** SAT 2006: 252-265

Exact MaxSAT Solvers Timeline



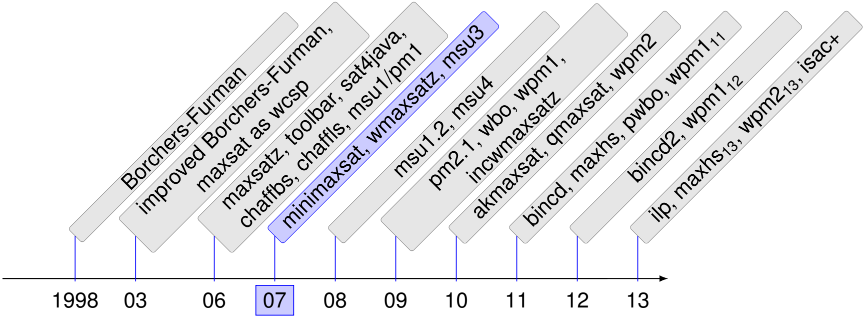
2006: **msu1/pm1** [Fu and Malik, 2006]

Contribution: first core-guided approach, clauses relaxed on demmand, multiple relaxing variables per clause

MaxSAT Evaluations: from 2008

Zhaohui Fu, Sharad Malik: **On Solving the Partial MAX-SAT Problem.** SAT 2006: 252-265

Exact MaxSAT Solvers Timeline

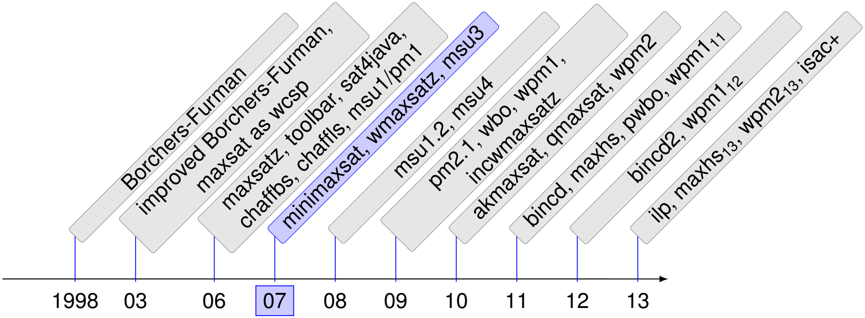


2007: **minimaxsat** [Heras et al., 2008]

Contribution: B&B on minisat with restricted maxsat resolution rule

MaxSAT Evaluations: 2007, 2008

Federico Heras, Javier Larrosa, Albert Oliveras: **MiniMaxSAT: An Efficient Weighted Max-SAT solver.** J. Artif. Intell. Res. (JAIR) 31: 1-32 (2008)



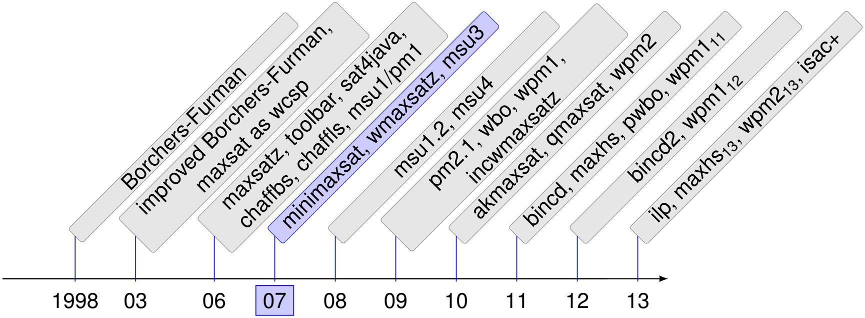
2007: **wmaxsatz** [Li et al., 2010]

Contribution: Weighted version of maxsatz

MaxSAT Evaluations: from 2007

Chu Min Li, Felip Manya, Nouredine Ould Mohamedou, Jordi Planes: **Resolution-based lower bounds in MaxSAT.** Constraints 15(4): 456-484 (2010)

Exact MaxSAT Solvers Timeline



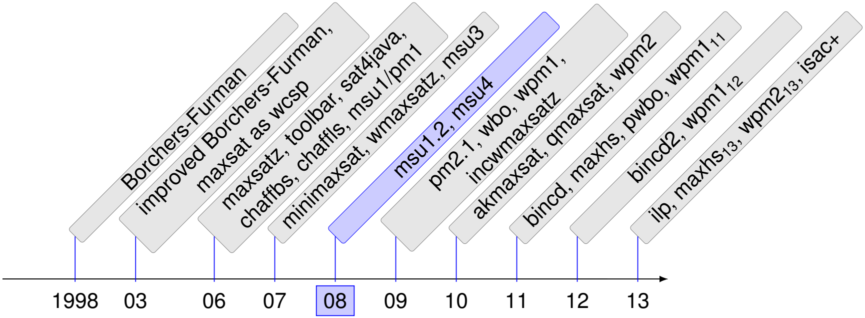
2007: **msu3** [Marques-Silva and Planes, 2007b]

Contribution: clauses relaxed on demmand, only one blocking variable

MaxSAT Evaluations: -

João Marques-Silva, Jordi Planes: **On Using Unsatisfiability for Solving Maximum Satisfiability.** CoRR abs/0712.1097 (2007)

Exact MaxSAT Solvers Timeline



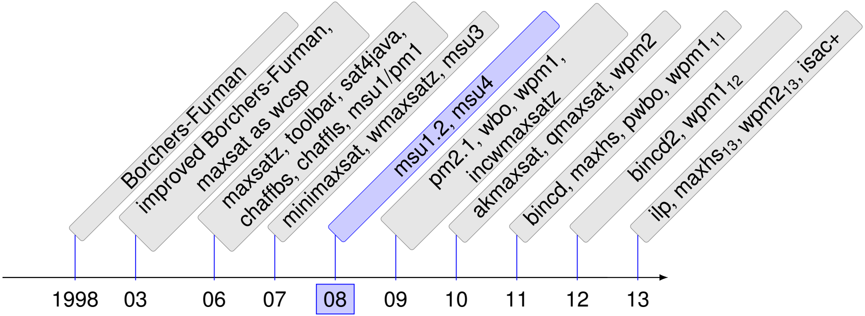
2008: **msu1.2 [Marques-Silva and Manquinho, 2008]**

Contribution: first core-guided Fu-Malik solver at maxsat evaluation

MaxSAT Evaluations: from 2008

João Marques-Silva, Vasco M. Manquinho: **Towards More Effective Unsatisfiability-Based Maximum Satisfiability Algorithms.** SAT 2008: 225-230

Exact MaxSAT Solvers Timeline

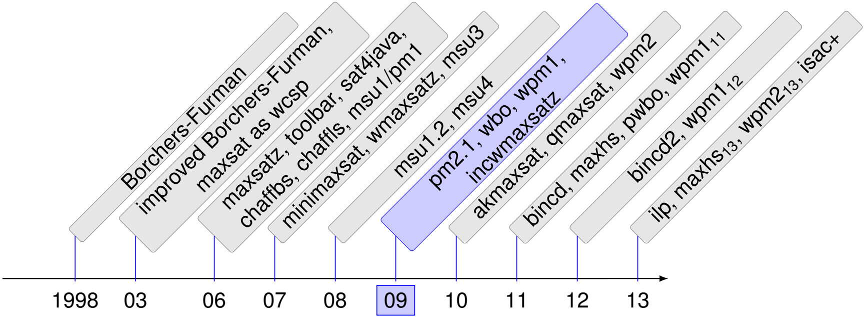


2008: **msu4 [Marques-Silva and Planes, 2008]**

Contribution: SAT-based solver alternating sat and unsat calls

MaxSAT Evaluations: 2008 João Marques-Silva, Jordi Planes: **Algorithms for Maximum Satisfiability using Unsatisfiable Cores**, DATE08, 2008, 408–413.

Exact MaxSAT Solvers Timeline



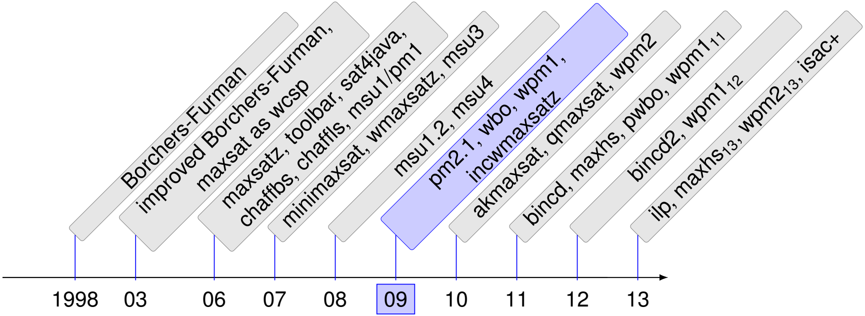
2009: **pm2.1 [Ansótegui et al., 2009]**

Contribution: covers for core-guided solvers

MaxSAT Evaluations: from 2009 to 2012

Carlos Ansotegui, Maria Luisa Bonet, Jordi Levy: **On Solving MaxSAT Through SAT.** CCIA 2009: 284-292

Exact MaxSAT Solvers Timeline



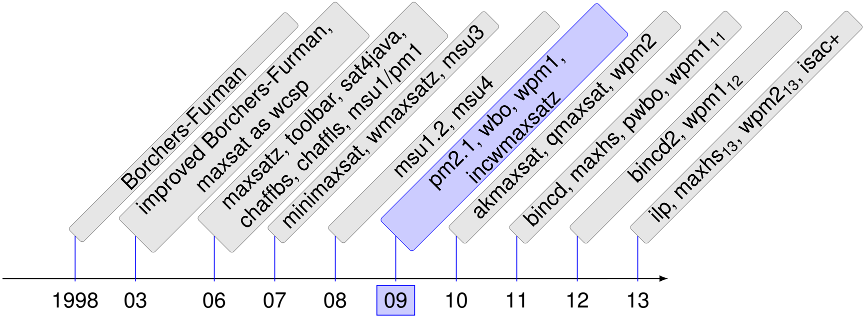
2009: **wpm1 [Ansotegui et al., 2009]**

Carlos Ansotegui, Maria Luisa Bonet, Jordi Levy: Solving (Weighted) Partial MaxSAT through Satisfiability Testing. SAT 2009: 427-440

Contribution: weighted version of core-guided Fu-Malik algorithm

MaxSAT Evaluations: from 2009

Exact MaxSAT Solvers Timeline



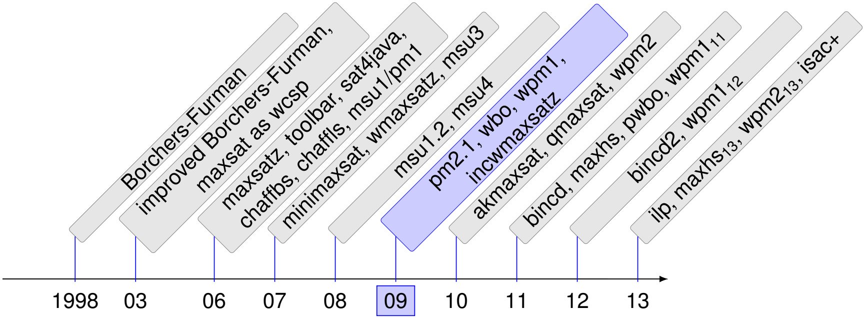
2009: **wbo [Marques-Silva and Planes, 2008]**

Contribution: weighted version of core-guided Fu-Malik algorithm

MaxSAT Evaluations: from 2009

João Marques-Silva, Jordi Planes: **Algorithms for Maximum Satisfiability using Unsatisfiable Cores**, DATE08, 2008, 408–413.

Exact MaxSAT Solvers Timeline



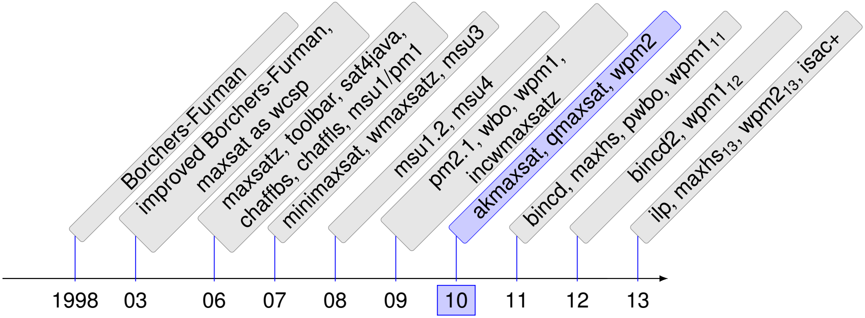
2009: **incwmaxsatz [Lin et al., 2008]**

Contribution: maxsatz plus property guarenteing the increment of lower bounds

MaxSAT Evaluations: from 2009

Han Lin and Kaile Su and Chu Min Li: **Within-problem Learning for Efficient Lower Bound Computation in Max-SAT Solving**, AAAI08, 2008, 351–356

Exact MaxSAT Solvers Timeline



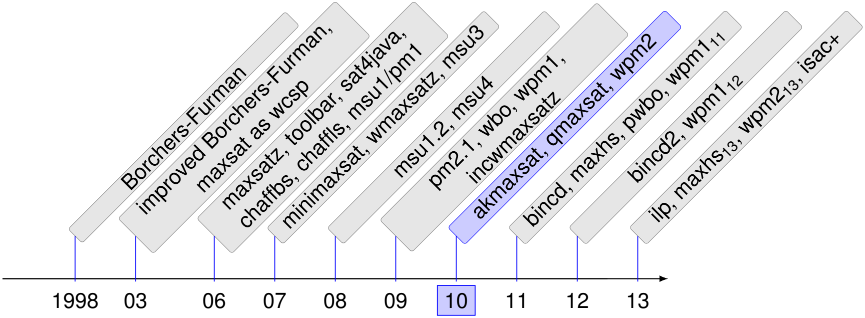
2010: **akmaxsat**

Contribution: improved version of wmaxsatz techniques

MaxSAT Evaluations: from 2010

A. Kuegel: **Improved exact solver for the weighted max-sat problem**.

Exact MaxSAT Solvers Timeline



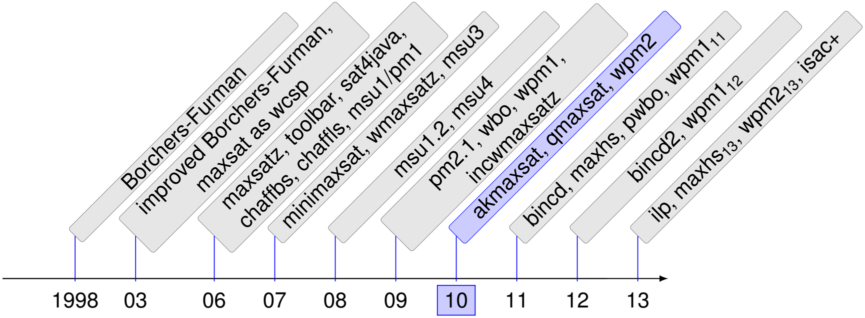
2010: **qmaxsat** [Koshimura et al., 2012]

Contribution: effcient model-guided solver for partial maxsat (as sat4java)

MaxSAT Evaluations: from 2010

Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, Ryuzo Hasegawa: **QMaxSAT: A Partial Max-SAT Solver.** JSAT 8(1/2): 95-100 (2012)

Exact MaxSAT Solvers Timeline

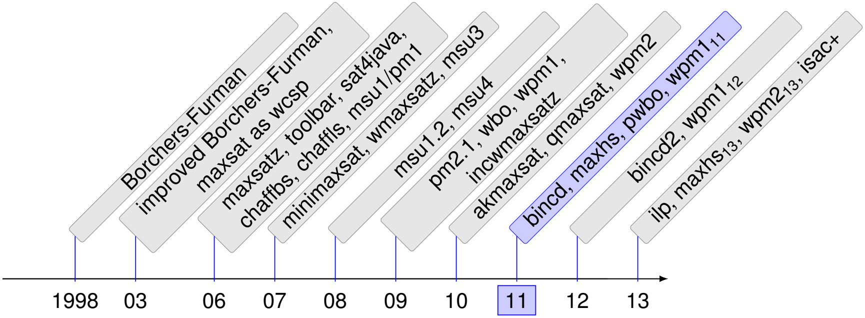


2010: **wpm2 [Ansotegui et al., 2010]**

Contribution: weighted version of *pm2*. Optimization oracle for new LB.

MaxSAT Evaluations: from 2010

Carlos Ansotegui, Maria Luisa Bonet, Jordi Levy: **A New Algorithm for Weighted Partial MaxSAT.** AAAI 2010



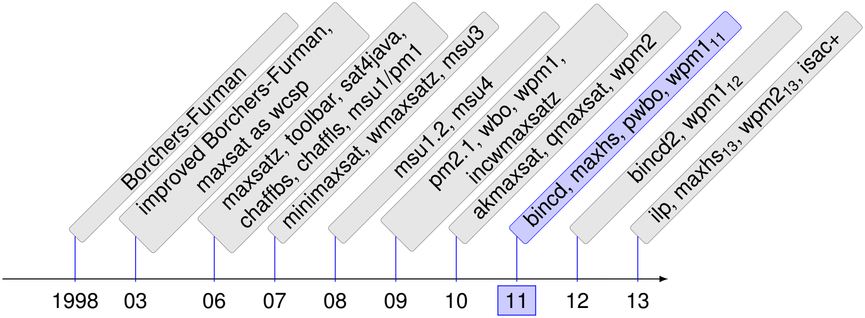
2011: **bincd [Heras et al., 2011]**

Contribution: core-guided binary search

MaxSAT Evaluations: -

Federico Heras, Antonio Morgado, Joao Marques-Silva: **Core-Guided Binary Search Algorithms for Maximum Satisfiability**, AAAI 2011

Exact MaxSAT Solvers Timeline



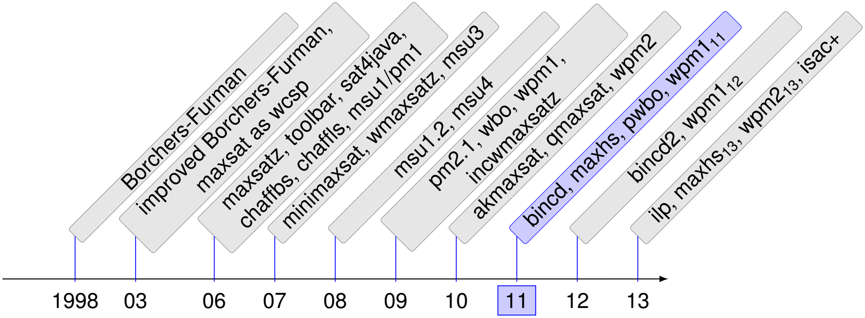
2011: **maxhs [Davies and Bacchus, 2011]**

Contribution: hybrid approach of SAT and OR

MaxSAT Evaluations: -

Jessica Davies, Fahiem Bacchus: **Solving MAXSAT by Solving a Sequence of Simpler SAT Instances.** CP 2011: 225-239

Exact MaxSAT Solvers Timeline



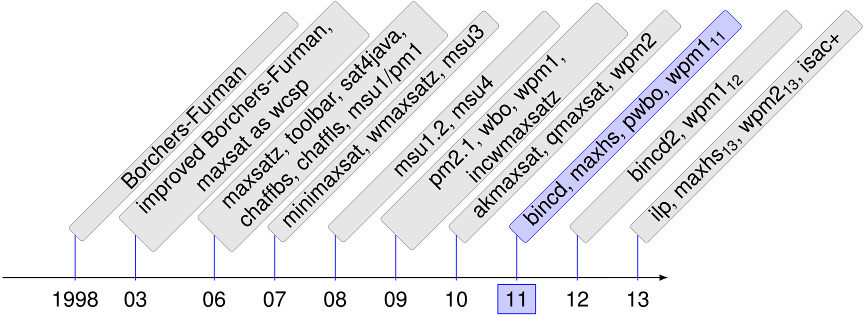
2011: **pwbo [Martins et al., 2012]**

Contribution: One thread for increasing LB, one thread for decreasing UB

MaxSAT Evaluations: from 2011

Ruben Martins, Vasco M. Manquinho, Inês Lynce: **Parallel search for maximum satisfiability.** AI Commun. 25(2): 75-95 (2012)

Exact MaxSAT Solvers Timeline



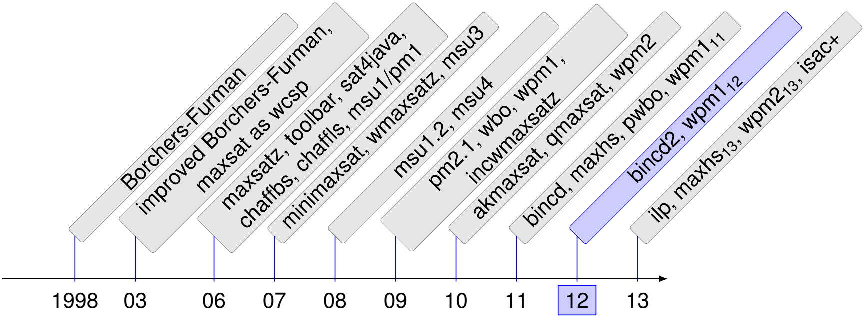
2011: **wpm1$_{11}$ [Ansótegui et al., 2012]**

Contribution: stratification approach

MaxSAT Evaluations: 2013

Carlos Ansotegui, Maria Luisa Bonet, Joel Gabas and Jordi Levy. **Improving SAT-Based Weighted MaxSAT Solvers.**

Exact MaxSAT Solvers Timeline



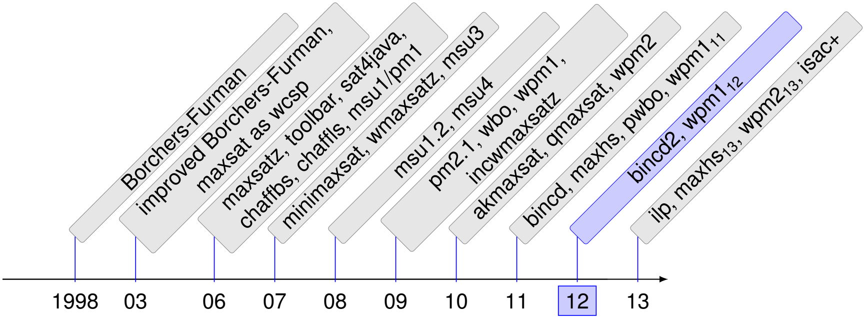
2012: **bincd2 [Morgado et al., 2012]**

Contribution: LB and UB computation improved. Hardening of soft clauses

MaxSAT Evaluations: -

Antonio Morgado, Federico Heras, Joao Marques-Silva: **Improvements to Core-Guided Binary Search for MaxSAT.** SAT 2012: 284-297

Exact MaxSAT Solvers Timeline



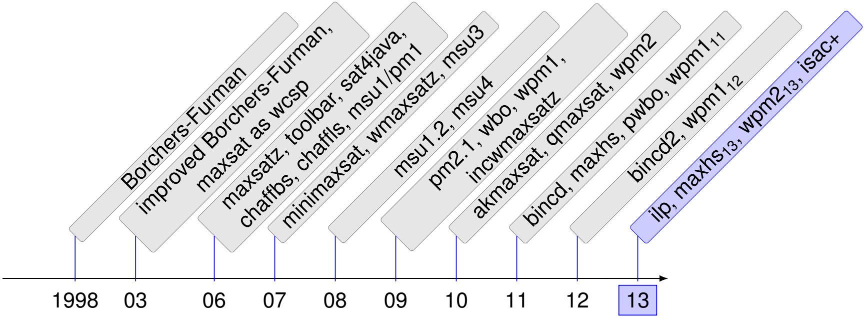
2012: **wpm1$_{12}$ [Ansótegui et al., 2012]**

Contribution: stratification apporach + diversity heuristic, symmetry breaking of blocking variables. Hardening of soft clauses

MaxSAT Evaluations: 2013

Carlos Ansotegui, Maria Luisa Bonet, Joel Gabas and Jordi Levy. **Improving SAT-Based Weighted MaxSAT Solvers.**

Exact MaxSAT Solvers Timeline



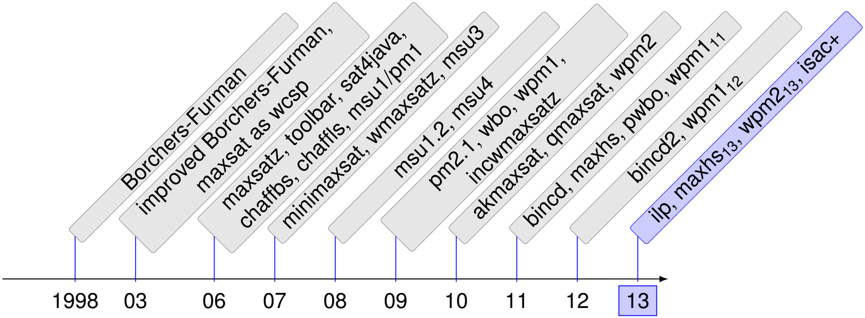
2013: **ilp [Ansotegui and Gabas, 2013]**

Contribution: MaxSat frontend for CPLEX

MaxSAT Evaluations: -

Carlos Ansotegui, Joel Gabás: **Solving (Weighted) Partial MaxSAT with ILP.** CPAIOR 2013: 403-409

Exact MaxSAT Solvers Timeline



2013:

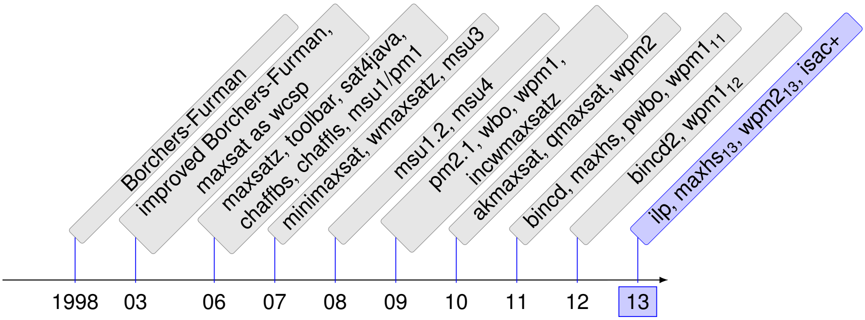
maxhs$_{13}$ [Davies and Bacchus, 2013a, Davies and Bacchus, 2013b]

Contribution: more constraints to OR. Postpone calls to OR.

MaxSAT Evaluations: 2013

Jessica Davies, Fahiem Bacchus: **Exploiting the Power of mip Solvers in maxsat.** SAT 2013: 166-181

Exact MaxSAT Solvers Timeline



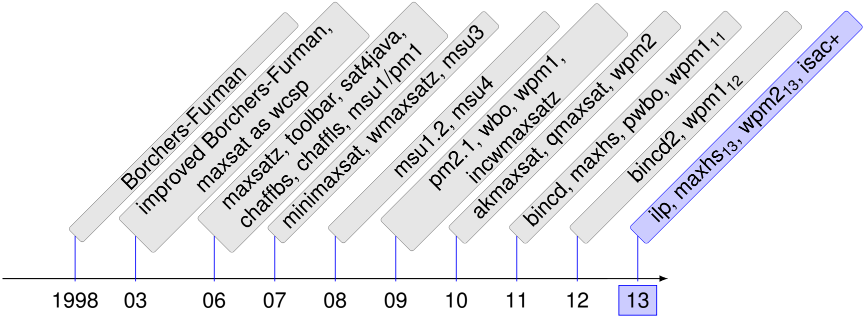
2013: **wpm2$_{13}$ [Ansótegui et al., 2013a]**

Contribution: cover optimization. Hardening of soft clauses

MaxSAT Evaluations: 2013

Carlos Ansotegui, Maria Luisa Bonet, Joel Gabas and Jordi Levy. **Improving WPM2 for (Weighted) Partial MaxSAT.**

Exact MaxSAT Solvers Timeline



2013: **isac+**

Contribution: Portfolio for MaxSAT

MaxSAT Evaluations: 2013

Carlos Ansotegui, Joel Gabas, Yuri Malitsky, Meinolf Selmann. **ISAC+ at MaxSAT Evaluation 2013**

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

MaxSAT Evaluation 2013: all categories.

	MS	WMS	PMS	WPMS
Random	*MaxSatz2013f* *ISAC+* *ckmax-small*	*ckmax-small* *ISAC+* *Maxsatz2013f*	*ISAC+* *WMaxSatz09* *WMaxSatz+*	*ISAC+* *WMaxSatz09* *WMaxSatz+*
Crafted	*ISAC+* *Maxsatz2013f* *ckmax-small*	*WMaxSatz+* *WMaxSatz09* *ckmaxsat-small*	*ISAC+* *ILP* *scip-maxsat*	*MaxHS* *ISAC+* *ILP*
Industrial	*pmifumax* *WPM1-2011* *ISAC+*	– – –	*ISAC+* *QMaxSAT2-mt* *MSUnCore*	*WPM1-2013* *ISAC+* *WPM2-2013*

Table: Ordered by solved instances.

MaxSAT Evaluation 2013: all categories.

	MS	WMS	PMS	WPMS
Random	*MaxSatz2013f* *ISAC+* *ckmax-small*	*ckmax-small* *ISAC+* *Maxsatz2013f*	*ISAC+* *WMaxSatz09* *WMaxSatz+*	*ISAC+* *WMaxSatz09* *WMaxSatz+*
Crafted	▲ *ahmaxsat* ▼ *ISAC+* ▼ *Maxsatz2013f*	*WMaxSatz+* *WMaxSatz09* ▲ *MaxSatz2013f*	*ISAC+* ▲ *QMaxSAT-m* ▲ *QMaxSAT2-mt*	*MaxHS* *ISAC+* *ILP*
Industrial	*pmifumax* *WPM1-2011* *ISAC+*	– – –	*ISAC+* *QMaxSAT2-mt* ▲ *WPM2-2013*	▲ *ISAC+* ▼ *WPM1-2013* *WPM2-2013*

Table: Ordered by mean ratio.

MSE 2013 results: crafted categories

	MS	WMS	PMS	WPMS
Crafted	*ahmaxsat* 86,3% - 146 *Maxsatz2013f* 85,8% - 155 *ckmax-small* 85,8% - 155 *WMaxSatz09* 85,3% - 154 *WMaxSatz+* 85,3% - 154	*WMaxSatz+* 59,8% - 79 *WMaxSatz09* 59,8% - 79 *Maxsatz2013f* 54,0% - 77 *ILP-2013* 53,0% - 61 *scip-maxsat* 46,8% - 56	*QMaxSAT-m* 71,2% - 285 *QMaxSAT2-mt* 71,1% - 272 *antom_seq1* 69,6% - 275 *MaxHS* 69,6% - 300 *ILP-2013* 66,4% - 327	*MaxHS* 89,0% - 330 *ILP-2013* 76,6% - 305 *WPM1-2013* 73,3% - 292 *pwbo2.3-wpms* 66,9% - 221 *scip-maxsat* 66,6% - 252

Table: Ordered by ratio (ratio - instances).

MSE 2013 results: all crafted instances

Crafted	Total
ILP-2013	61,1% - 723
MaxHS	59,9% - 670
*WMaxSatz*09	59,2% - 745
scip-maxsat	55,1% - 657
QMaxSAT2-mt	47,2% - 481
WPM2-2013	46,3% - 521

Table: Ordered by ratio (ratio - instances).

MSE 2013 results: industrial categories

	MS	WMS	PMS	WPMS
Industrial	*pmifumax*	-	*QMaxSAT2-mt*	*WPM1*-2013
	84,5% - 39	-	84,4% - 534	77,0% - 344
	WMP1-2011	-	*WPM2*-2013	*WPM2*-2013
	82,5% - 37	-	78,6% - 485	73,4% - 318
	optimax	-	*optimax-ni*	*MSUnCore*
	76,5% - 31	-	78,5% - 484	66,0% - 266
	optimax-ni	-	*QMaxSAT-m*	*pwbo2.3-wpms*
	75,5% - 30	-	76,0% - 475	56,3% - 265
	wbo2.1-cnf	-	*MSUnCore*	*MasHS*
	73,0% - 27	-	75,8% - 497	53,1% - 255

Table: Ordered by ratio (ratio - instances).

MSE 2013 results: all industrial instances

Industrial	Total
WPM2-2013	75,0% - 820
MSUnCore	71,4% - 784
optimax-ni	70,2% - 671
QMaxSAT2-mt	68,7% - 640
WPM1-2013	65,0% - 743
pwbo2.3	63,0% - 686
MaxHS	59,7% - 719

Table: Ordered by ratio (ratio - instances).

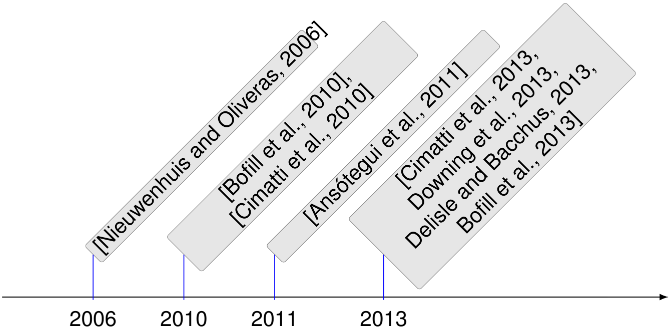
MSE 2013 results: all crafted and industrial instances

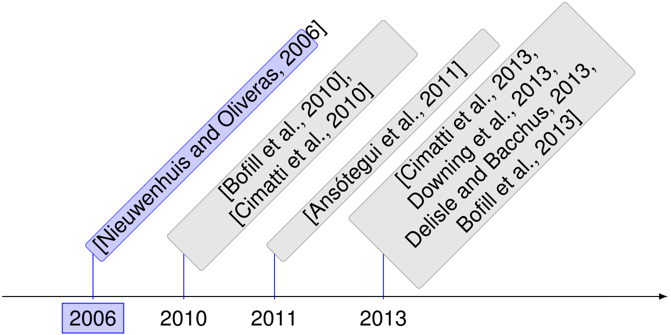
Crafted and Industrial	Total
WPM2-2013	61,5% - 1341
MaxHS	59,8% - 1389
QMaxSAT2-mt	58,6% - 1121

Table: Ordered by ratio (ratio - instances).

Outline

1. The MaxSAT problem
2. Modeling problems as MaxSAT
3. SAT-based MaxSAT solvers
4. Results at MaxSAT evaluation 2013
5. Extensions of MaxSAT

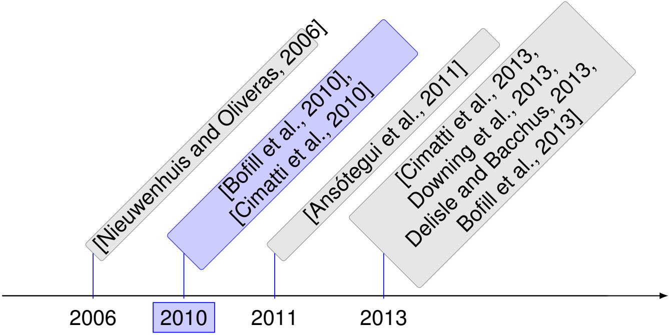

[Nieuwenhuis and Oliveras, 2006]
[Bofill et al., 2010],
[Cimatti et al., 2010]
[Ansótegui et al., 2011]
[Cimatti et al., 2013,
Downing et al., 2013,
Delisle and Bacchus, 2013,
Bofill et al., 2013]
2006
2010
2011
2013



2006: **[Nieuwenhuis and Oliveras, 2006]**

Contribution: model-guided search

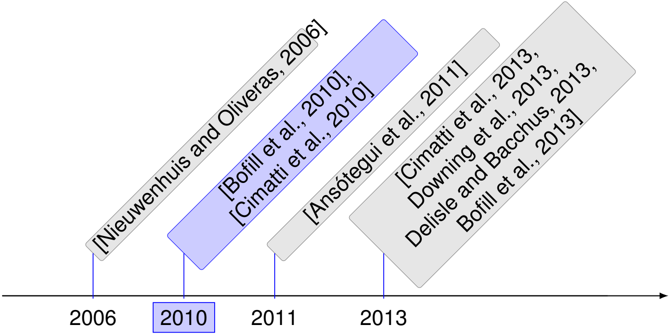
Robert Nieuwenhuis, Albert Oliveras: **On SAT Modulo Theories and Optimization Problems.** SAT 2006: 156-169



2010: **[Bofill et al., 2010]**

Contribution: binary search on the domain of the variable to optimize

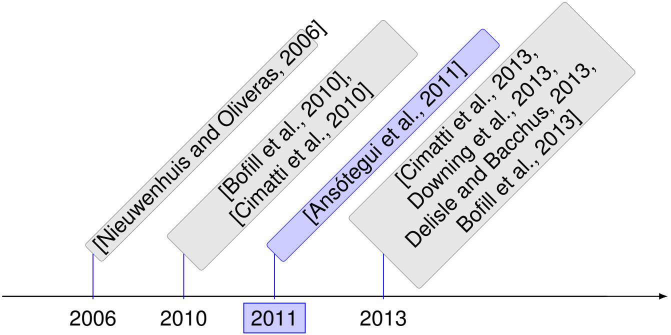
Miquel Bofill, Josep Suy, Mateu Villaret: **A System for Solving Constraint Satisfaction Problems with SMT.** SAT 2010: 300-305



2010: **[Cimatti et al., 2010]**

Contribution: theory of costs. binary and model-guided search.

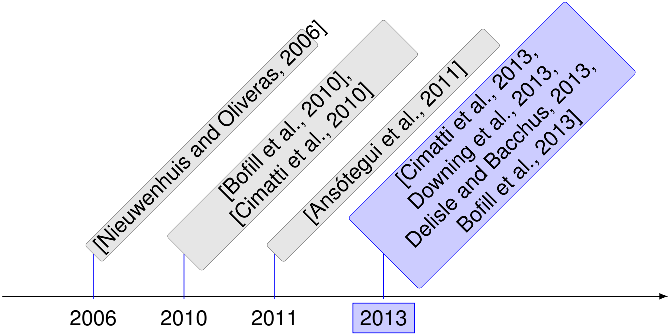
Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, Cristian Stenico: **Satisfiability Modulo the Theory of Costs: Foundations and Applications.** TACAS 2010: 99-113



2010: **[Ansótegui et al., 2011]**

Contribution: core-guided approach for intensional WCSPs (as WSMT)

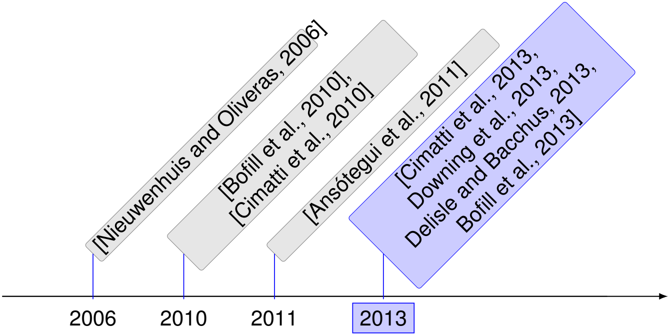
C. Ansotegui, M. Bofill, M. Palahi, J. Suy, M. Villaret: **A Proposal for Solving Weighted CSPs with SMT** ModRef 2011: 5-19



2013: **[Cimatti et al., 2013]**

Contribution: cyclic interaction of a lazy SMT and a MaxSAT solver

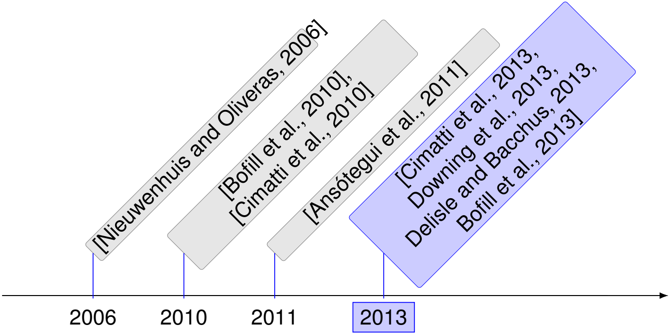
Alessandro Cimatti, Alberto Griggio, Bastiaan Joost Schaafsma, Roberto Sebastiani: **A Modular Approach to MaxSAT Modulo Theories.** SAT 2013: 150-165



2013: **[Downing et al., 2013]**

Contribution: core-guided approach for Constraint Programming

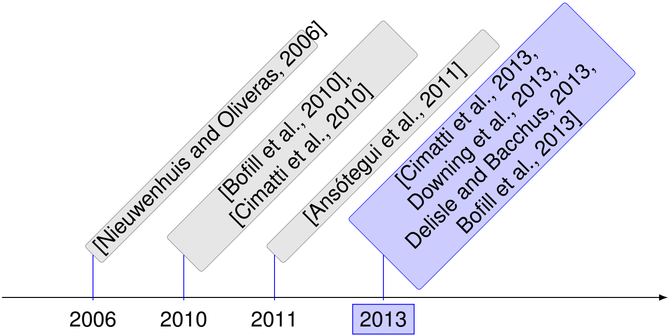
Nicholas Downing, Thibaut Feydy, Peter J. Stuckey: **Unsatisfiable Cores for Constraint Programming.** CoRR abs/1305.1690 (2013)



2013: **[Delisle and Bacchus, 2013]**

Contribution: MaxHS-like approach for extensional WCSPs.

Erin Delisle and Fahiem Bacchus: **Solving weighted CSPs by Successive Relaxations.**, CP 2013



2013: **[Bofill et al., 2013]**

Contribution: use of shared BDDs to represent objective function

M. Bofill and M. Palahi and J. Suy and M. Villaret: **Boosting Weighted CSP Resolution with Shared BDDs**, ModRef 2013

- **SAT Modulo Theories (SMT)**

  Determine satisfiability of a first order formula F w.r.t. a background theory T:

$$A \wedge (B \vee x + 3 < y) \wedge x \geq y$$

  Boolean model:
  $A = \mathit{true}, B = \mathit{true}, (x + 3 < y) = \mathit{false}, (x \geq y) = \mathit{true}$
  if T is the theory of Linear Integer Arithmetic.

- Most common **SMT-solvers** use **SAT-solver + T-solver**
- SMT well suited for CSP solving: see fzn2smt in previous Minizinc challenges

- **SAT Modulo Theories (SMT)**

  Determine satisfiability of a first order formula F w.r.t. a background theory T:

  $$A \wedge (B \vee x + 3 < y) \wedge x \geq y$$

  Boolean model:
  $A = \mathit{true}, B = \mathit{true}, (x + 3 < y) = \mathit{false}, (x \geq y) = \mathit{true}$
  if T is the theory of Linear Integer Arithmetic.
- Most common **SMT-solvers** use **SAT-solver + T-solver**
- SMT well suited for CSP solving: see fzn2smt in previous Minizinc challenges

- **SAT Modulo Theories (SMT)** Determine satisfiability of a first order formula F w.r.t. a background theory T:

$$A \wedge (B \vee x + 3 < y) \wedge x \geq y$$

  Boolean model:
  $A = true, B = true, (x + 3 < y) = false, (x \geq y) = true$
  if T is the theory of Linear Integer Arithmetic.
- Most common **SMT-solvers** use **SAT-solver + T-solver**
- SMT well suited for CSP solving: see fzn2smt in previous Minizinc challenges

SAT Modulo Theories

- **SAT Modulo Theories (SMT)**
- Most common **SMT-solvers** use **SAT-solver + T-solver**
  1. The SAT solver finds a propositional model,

     $$A = true, B = true, (x + 3 < y) = true, (x \geq y) = true$$

  2. Then asks T-solver whether current assignment for literals falling into theory T is consistent with (first-order) theory T,

     Is $x + 3 < y, x \geq y$ consistent with
     Linear Integer Arithmetic?

     Obs.: T-solver works with conjunctions (set of literals)
  3. If inconsistency, backtrack.
- SMT well suited for CSP solving: see fzn2smt in previous Minizinc challenges

State-of-the-Art in encoding CSP into SMT

Objective

Provide a generic tool for efficiently solving CSP with SMT.

- Many attempts, more o less generic, to translate CSP into SAT: SUGAR, SPEC2SAT, CSP2SAT4J, FznTini, etc.
- No attempts to make a generic translation of CSP into SMT.
- First attempt was `Simply`, a declarative programming system for easy modeling and solving of CSP´s.
- Generic attempt: `fzn2smt`. A system for solving MiniZinc (standard language) instances using SMT.

MiniZinc

MiniZinc is a medium-level constraint modeling language to express constraint problems easily. It aims be adopted as a standard by the Constraint Programming community.

FlatZinc

FlatZinc is a low-level solver input language that is the target language for MiniZinc.

- The MiniZinc to FlatZinc translation (Flattening): list comprehension unrolling, fixed array accesses replacement, normalization of Boolean expressions, sub-expression replacement, etc.

The Balanced Academic Curriculum Problem (BACP) consists in assigning courses to teaching periods satisfying prerequisites and balancing students' load.

Example of MiniZinc: BACP

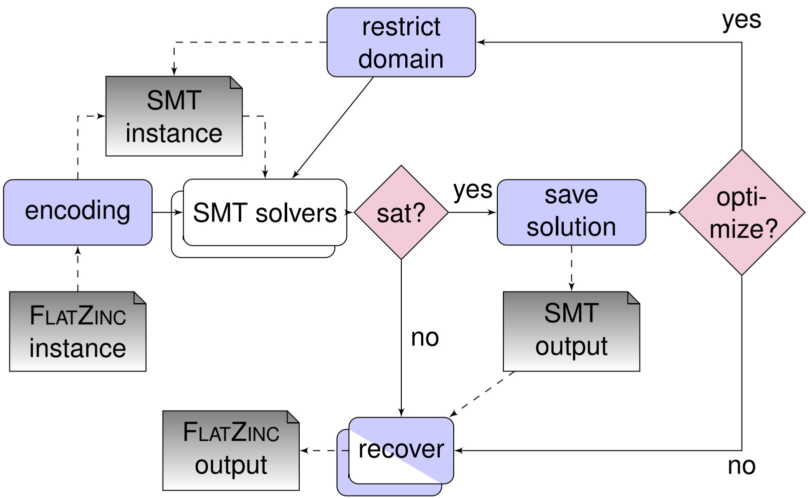
```
include "globals.mzn";
int: n_courses=9; int: n_periods=3;
int: load_per_period_lb=2; int: load_per_period_ub=5;
int: courses_per_period_lb=7; int: courses_per_period_ub=13;
% 1=Mat. 2=Cal. 3=Algebra 4=Alg. 5=Alg. II 6=Prog. 7=Prog.II
% 8= DDBB 9=Ph.
array [1..n_courses] of int: course_load=[3,4,3,1,4,1,4,3,2];
set of int: courses = 1..n_courses;
set of int: periods = 1..n_periods;
% Variables
array [courses] of var periods: course_period;
array [periods, courses] of var 0..1: x;
array [periods] of var load_per_period_lb..load_per_period_ub: load;
var load_per_period_lb..load_per_period_ub: objective;

constraint course_period[2] < course_period[1]; ...
constraint forall(p in periods) (
    forall(c in courses) (x[p,c] = bool2int(course_period[c] = p)) /\
    sum(i in courses) (x[p,i]) >= courses_per_period_lb /\
    sum(i in courses) (x[p,i]) <= courses_per_period_ub /\
    load[p] = sum(c in courses) (x[p,c] * course_load[c]) /\
    load[p] >= load_per_period_lb /\
    load[p] <= objective
);
constraint forall(p in 0..n_periods-1) (
    let {var int: l = sum(c in courses) (bool2int(course_period[c] > p) * course_load[c])} in
        l >= (n_periods-p) * load_per_period_lb /\
        l <= (n_periods-p) * objective
    );
solve :: minimize objective;
output [show(c) ++ "-" ++ show(course_period[c]) ++ "\t" | c in courses ] ++ [show(objective)
```

Example of FlatZinc: BACP

```
array [1..9] of int: course_load = [3, 4, 3, 1, 4, 1, 4, 3, 2];
var bool: BOOL____00001 :: is_defined_var :: var_is_introduced;
...
var 0..1: INT____00002 :: is_defined_var :: var_is_introduced;
...
array [1..9] of var 1..3: course_period :: output_array([1..9]);
array [1..3] of var 7..13: load;
var 7..13: objective :: output_var;
array [1..27] of var 0..1: x;
constraint bool2int(BOOL____00001, INT____00002) :: defines_var(INT____00002);
...
constraint bool2int(BOOL____00035, INT____00036) :: defines_var(INT____00036);
constraint int_eq_reif(course_period[1], 1, BOOL____00037) :: defines_var(BOOL____00037);
...
constraint int_eq_reif(course_period[9], 3, BOOL____00089) :: defines_var(BOOL____00089);
constraint int_le(load[1], objective);
constraint int_le(load[2], objective);
constraint int_le(load[3], objective);
constraint int_lin_eq([1, -3, -4, -3, -1, -4, -1, -4, -3, -2],
[load[1], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x[9]], 0);
...
constraint int_lin_le([-3, -4, -3, -1, -4, -1, -4, -3, -2],
[INT____00002, INT____00004, INT____00006, INT____00008, INT____00010, INT____00012,
INT____00014, INT____00016, INT____00018], -14);
...
constraint int_lt_reif(1, course_period[1], BOOL____00001) :: defines_var(BOOL____00001);
...
constraint int_lt_reif(2, course_period[9], BOOL____00035) :: defines_var(BOOL____00035);
solve  :: seq_search([int_search([objective])] minimize objective;
```

fzn2smt Characteristics

- Complete FLATZINC support.
  - Data types: integers, Booleans and floats.
  - Data structures: one dimensional arrays and sets of all basic types.
  - Constraints: constraints over integers, Booleans, floats, arrays and sets.
  - Solve items: satisfaction or minimization/maximization of an integer variable.
- High performance.
- Automatic determination of the theory.
- Choosable optimization approach.
- Possibility of using different SMT solvers (Yices, Z3, Barcelogic, . . . with or without APIs).
- Comparison with other solvers.



The compiling and solving process of fzn2smt.

State-of-the-Art in encoding WCSP into Weighted SMT

Objective

To develop a system supporting WCSP and metaconstraints, and to solve them using SMT.

- There exist several systems for specification and solving of extensional WCSP.
- Systems supporting the intensional modeling of WCSP do not exist.
- Need systems supporting metaconstraints, i.e., constraints on (soft) constraints. Helpful in the modeling process.
- Solving using Optimization SMT or WSMT.

- Simply is a declarative programming system for easy modelling of CSP and solving using SMT. It is similar to MiniZinc without reals, sets variables and some constraints.

- WSimply is an extension of Simply that supports the intensional modeling of WCSP and metaconstraints.

BACP encoded into `Simply`

```
Problem:bacp
 Data
    int n_courses; int n_periods;
    int load_per_period_lb; int load_per_period_ub;
    int courses_per_period_lb; int courses_per_period_ub;
    int course_load[n_courses]; int n_prereqs; int prereqs[n_prereqs,2];
 Domains
    Dom dperiods=[1..n_periods];
    Dom dload=[load_per_period_lb..load_per_period_ub];
    Dom dcourses=[courses_per_period_lb..courses_per_period_ub];
 Variables
    IntVar course_period[n_courses]::dperiods;
    IntVar period_load[n_periods]::dload;
    IntVar period_courses[n_periods]::dcourses;
 Constraints
     Forall(p in [1..n_periods]) {
       Count([course_period[c]|c in [1..n_courses]],p,period_courses[p]);
     };
     Forall(p in [1..n_periods]) {
       Sum([If_Then_Else(course_period[c]=p)(course_load[c])(0)
            |c in [1..n_courses]],period_load[p]);
     };
     Forall(np in [1..n_prereqs]) {
       course_period[prereqs[np,1]]<=course_period[prereqs[np,2]];
     };
```

- **Weighted Constraints:** `(constraint)@{weight}`
- **Degree of Violation:** `(constraint)@{expr}`
- **Labeled Constraints:** `#label:(constraint)@{expr}`
- **Undefined weights:** `#label:(constraint)@{_}`

Weighted constraints examples

$$(P_{Calculus} < P_{Mathematics})\texttt{@\{1\}}$$
$$(P_{Algebra} < P_{Mathematics})\texttt{@\{}\,P_{Algebra} - P_{Mathematics} + 1\,\texttt{\}}$$
$$\#P:(P_{Prog.} < P_{Prog.II})\texttt{@\{1\}}$$
$$(P_{Alg.I} < P_{Alg.II})\texttt{@\{\_\}}$$

We consider the following metaconstraints:

- Priority between soft-constraints.

Priority metaconstraints examples

$\#A : (P_{Calculus} < P_{Mathematics})$ `@ {1}`
$\#B : (P_{Algebra} < P_{Mathematics})$ `@ {_}`
$\#C : (P_{Prog.} < P_{Prog.II})$ `@ {_}`
$\#D : (P_{Alg.I} < P_{Alg.II})$ `@ {_}`
$\#E : (P_{Prog.II} < P_{DDBB})$ `@ {_}`

samepriority([A,B])
priority([B,C])
priority(D,B,3)
multilevel([A,B,C,D],[E])

We consider the following metaconstraints:

- Priority between soft-constraints.
- Homogeneity in the amount of soft constraints violation between groups.
  - `atLeast(List,p)`
  - `homogeneousAbsoluteWeight(ListOfList,v)`
  - `homogeneousAbsoluteNumber(ListOfList,v)`
  - `homogeneousPercentWeight(ListOfList,v)`
  - `homogeneousPercentNumber(ListOfList,v)`

Homogeneity metaconstraints examples

$\#A : (P_{Calculus} < P_{Mathematics})$`@{1}`
$\#B : (P_{Algebra} < P_{Mathematics})$`@{1}`
$\#C : (P_{Prog.} < P_{Prog.II})$`@{1}` $\#D : (P_{Prog.II} < P_{DDBB})$`@{3}`

homogeneousAbsoluteNumber([[A,B][C,D]],1)

`WSimply` Metaconstraints

We consider the following metaconstraints:

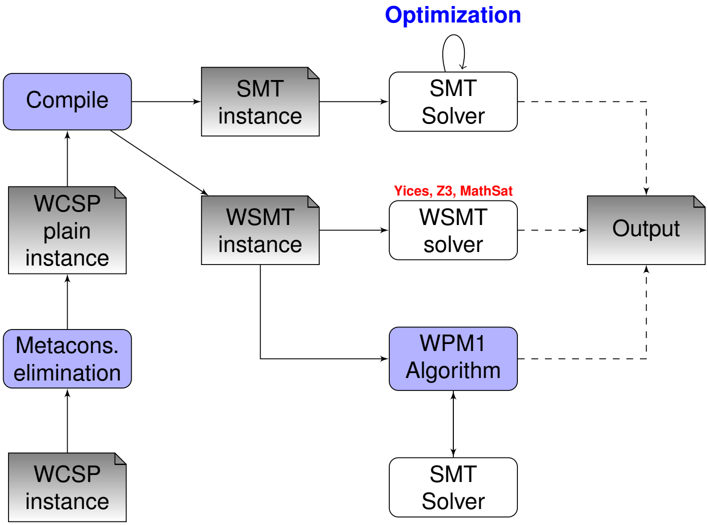
- Priority between soft-constraints.
- Homogeneity in the amount of soft constraints violation between groups.
- Dependence.

Dependence metaconstraint example

$\#A : (P_{Calculus} < P_{Mathematics})$`@ { 1 }`

`(Not A) Implies` $(P_{Calculus} = P_{Mathematics})$

Weighted Simply



Concluding remarks

- SAT/SMT efficient for a decision problem?,
  try MaxSAT/MaxSMT for the optimization variant
- SAT-based MaxSAT solvers best performance on industrial instances
- Work on efficient encodings of Card and PB constraints
- Work on efficient PB or SMT (with LIA theory) solvers
- Branch and bound solvers for industrial instances, why not?
- Learn more from OR techniques
- MaxSMT is a promising area
- So far, all recent research effort on *Exact* MaxSAT solvers

Thanks!

Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., and Mayer-Eichberger, V. (2012).
A new look at bdds for pseudo-boolean constraints.
J. Artif. Intell. Res. (JAIR), 45:443–480.

Alsinet, T., Manyà, F., and Planes, J. (2003).
Improved branch and bound algorithms for Max-SAT.
In *Proceedings of the 6th International Conference on the Theory and Applications of Satisfiability Testing*.

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., and Villaret, M. (2011).
A proposal for solving weighted csps with smt.
In *Proceedings of the 10th International Workshop on Constraint Modelling and Reformulation (ModRef 2011)*, pages 5–19.

Ansótegui, C., Bonet, M. L., Gabàs, J., and Levy, J. (2012).
Improving sat-based weighted maxsat solvers.
In *Proc. of the 18th Int. Conf. on Principles and Practice of Constraint Programming (CP'12)*, pages 86–101.

Ansótegui, C., Bonet, M. L., Gabàs, J., and Levy, J. (2013a).
Improving wpm2 for (weighted) partial maxsat.
In *CP*.

Ansótegui, C., Bonet, M. L., and Levy, J. (2009).
On solving MaxSAT through SAT.
In *Proc. of the 12th Int. Conf. of the Catalan Association for Artificial Intelligence (CCIA'09)*, pages 284–292.

Ansotegui, C., Bonet, M. L., and Levy, J. (2009).
Solving (weighted) partial maxsat through satisfiability testing.
In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, pages 427–440.

References II

Ansotegui, C., Bonet, M. L., and Levy, J. (2010).
A new algorithm for weighted partial maxsat.
In *Proc. the 24th National Conference on Artificial Intelligence (AAAI'10)*.

Ansótegui, C., Bonet, M. L., and Levy, J. (2013b).
Sat-based maxsat algorithms.
Artif. Intell., 196:77–105.

Ansotegui, C. and Gabas, J. (2013).
Solving maxsat with mip.
In *CPAIOR*.

Asín, R., Nieuwenhuis, R., Oliveras, A., and Rodríguez-Carbonell, E. (2011).
Cardinality Networks: a theoretical and empirical study.
Constraints, 16(2):195–221.

Bailleux, O. and Boufkhad, Y. (2003).
Efficient CNF Encoding of Boolean Cardinality Constraints.
In Rossi, F., editor, *Principles and Practice of Constraint Programming, 9th International Conference, CP '03*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer.

Bailleux, O., Boufkhad, Y., and Roussel, O. (2006).
A Translation of Pseudo Boolean Constraints to SAT.
Journal on Satisfiability, Boolean Modeling and Computation, JSAT, 2(1-4):191–200.

Bailleux, O., Boufkhad, Y., and Roussel, O. (2009).
New Encodings of Pseudo-Boolean Constraints into CNF.
In Kullmann, O., editor, *12th International Conference on Theory and Applications of Satisfiability Testing, SAT '09*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer.

References III

Berre, D. L. (2006).
Sat4j, a satisfiability library for java.
www.sat4j.org.

Bofill, M., Palahi, M., Suy, J., and Villaret, M. (2013).
Boosting Weighted CSP Resolution with Shared BDDs.
In *Proceedings of the 12th International Workshop on Constraint Modelling and Reformulation*, pages 57–73.

Bofill, M., Suy, J., and Villaret, M. (2010).
A system for solving constraint satisfaction problems with smt.
In *13th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, volume 6175 of *Lecture Notes in Computer Science*, pages 300–305. Springer.

Borchers, B. and Furman, J. (1998).
A two-phase exact algorithm for max-sat and weighted max-sat problems.
J. Comb. Optim., 2(4):299–306.

Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., and Stenico, C. (2010).
Satisfiability modulo the theory of costs: Foundations and applications.
In *TACAS*, pages 99–113.

Cimatti, A., Griggio, A., Schaafsma, B. J., and Sebastiani, R. (2013).
A modular approach to maxsat modulo theories.
In *SAT*, pages 150–165.

Codish, M. and Zazon-Ivry, M. (2010).
Pairwise cardinality networks.
In Clarke, E. M. and Voronkov, A., editors, *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 154–172. Springer.

Davies, J. and Bacchus, F. (2011).
Solving maxsat by solving a sequence of simpler sat instances.
In *Proc. of the 17th Int. Conf. on Principles and Practice of Constraint Programming (CP'11)*, pages 225–239.

Davies, J. and Bacchus, F. (2013a).
Exploiting the power of mip solvers in maxsat.
In *SAT*, pages 166–181.

Davies, J. and Bacchus, F. (2013b).
Postponing optimization to speed up maxsat solving.
In *CP*.

de Givry, S., Larrosa, J., Meseguer, P., and Schiex, T. (2003).
Solving max-sat as weighted csp.
In *CP*, pages 363–376.

Delisle, E. and Bacchus, F. (2013).
Solving weighted csps by successive relaxations.
In *CP*.

Downing, N., Feydy, T., and Stuckey, P. J. (2013).
Unsatisfiable cores for constraint programming.
CoRR, abs/1305.1690.

Eén, N. and Sörensson, N. (2006a).
Translating Pseudo-Boolean Constraints into SAT.
Journal on Satisfiability, Boolean Modeling and Computation, 2:1–26.

Eén, N. and Sörensson, N. (2006b).
Translating pseudo-boolean constraints into SAT.
JSAT, 2(1-4):1–26.

Fu, Z. and Malik, S. (2006).
On solving the partial max-sat problem.
In *Proc. of the 9th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'06)*, pages 252–265.

Heras, F., Larrosa, J., and Oliveras, A. (2008).
Minimaxsat: An efficient weighted max-sat solver.
J. Artif. Intell. Res. (JAIR), 31:1–32.

Heras, F., Morgado, A., and Marques-Silva, J. (2011).
Core-guided binary search algorithms for maximum satisfiability.
In *Proc. the 25th National Conference on Artificial Intelligence (AAAI'11)*.

Honjyo, K. and Tanjo, T. (2012).
Shinmaxsat.
A Weighted Partial Max-SAT solver inspired by MiniSat+, Information Science and Technology Center, Kobe University.

Koshimura, M., Zhang, T., Fujita, H., and Hasegawa, R. (2012).
Qmaxsat: A partial max-sat solver.
JSAT, 8(1/2):95–100.

Larrosa, J., Heras, F., and de Givry, S. (2008).
A logical approach to efficient max-sat solving.
Artif. Intell., 172(2-3):204–233.

Li, C. M., Manyà, F., Mohamedou, N. O., and Planes, J. (2010).
Resolution-based lower bounds in maxsat.
Constraints, 15(4):456–484.

Li, C. M., Manyà, F., and Planes, J. (2007).
New inference rules for Max-SAT.
J. Artif. Intell. Res. (JAIR), 30:321–359.

Lin, H., Su, K., and Li, C. M. (2008).
Within-problem learning for efficient lower bound computation in Max-SAT solving.
In *Proc. the 23th National Conference on Artificial Intelligence (AAAI'08)*, pages 351–356.

Manquinho, V., Marques-Silva, J., and Planes, J. (2009).
Algorithms for weighted boolean optimization.
In *Proc. of the 12th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'09)*, pages 495–508.

Marques-Silva, J. and Manquinho, V. M. (2008).
Towards more effective unsatisfiability-based maximum satisfiability algorithms.
In *Proc. of the 11th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'08)*, pages 225–230.

Marques-Silva, J. and Planes, J. (2007a).
On using unsatisfiability for solving maximum satisfiability.
CoRR, abs/0712.1097.

Marques-Silva, J. and Planes, J. (2007b).
On using unsatisfiability for solving maximum satisfiability.
CoRR, abs/0712.1097.

Marques-Silva, J. and Planes, J. (2008).
Algorithms for maximum satisfiability using unsatisfiable cores.
In *Proc. of the Conf. on Design, Automation and Test in Europe (DATE'08)*, pages 408–413.

Martins, R., Manquinho, V. M., and Lynce, I. (2012).
Parallel search for maximum satisfiability.
AI Commun., 25(2):75–95.

Morgado, A., Heras, F., and Marques-Silva, J. (2012).
Improvements to core-guided binary search for maxsat.
In *Proc. of the 15th Int. Conf. on Theory and Applications of Satisfiability Testing (SAT'12)*, pages 284–297.

Nieuwenhuis, R. and Oliveras, A. (2006).
On sat modulo theories and optimization problems.
In *SAT*, pages 156–169.

Papadimitriou, C. H. (1994).
Computational complexity.
Addison-Wesley.

Sanchez, M., Bouveret, S., Givry, S. D., Heras, F., JÃ©gou, P., Larrosa, J., Ndiaye, S., Rollon, E., Schiex, T., Terrioux, C., Verfaillie, G., and Zytnicki, M. (2008).
Max-CSP competition 2008: toulbar2 solver description.

Sinz, C. (2005).
Towards an optimal CNF encoding of boolean cardinality constraints.
In v. Beek, P., editor, *Principles and Practice of Constraint Programming, 11th International Conference, CP '05*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer.

Warners, J. P. (1998).
A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form.
Information Processing Letters, 68(2):63–69.