

SAT Solving and Applications

Introduction to MaxSAT Solving

Prof. Dr. Wolfgang Küchlin
Rouven Walter, M.Sc.

University of Tübingen

25 July 2015



Contents

① Boolean Optimization

- MaxSAT
- Pseudo-Boolean Optimization (PBO)
- Conversion between PBO and MaxSAT

② Example Applications

- Configuration and Re-Configuration
- Minimal Vertex Cover

③ Techniques

- Cardinality Constraints
- Pseudo-Boolean Constraints

④ Practical Algorithms

- Branch & Bound (not covered)
- Iterative SAT Solving
- Core-Guided Algorithms (not covered)

What is MaxSAT?

Example

$$x_6 \vee x_2$$

$$\neg x_6 \vee x_2$$

$$\neg x_2 \vee x_1$$

$$\neg x_1$$

$$\neg x_6 \vee x_8$$

$$x_6 \vee \neg x_8$$

$$x_2 \vee x_4$$

$$\neg x_4 \vee x_5$$

$$x_7 \vee x_5$$

$$\neg x_7 \vee x_5$$

$$\neg x_5 \vee x_3$$

$$\neg x_3$$

Is this clause set satisfiable?

What is MaxSAT?

No!

$$x_6 \vee x_2$$

$$\neg x_6 \vee x_2$$

$$\neg x_2 \vee x_1$$

$$\neg x_1$$

$$\neg x_6 \vee x_8$$

$$x_6 \vee \neg x_8$$

$$x_2 \vee x_4$$

$$\neg x_4 \vee x_5$$

$$x_7 \vee x_5$$

$$\neg x_7 \vee x_5$$

$$\neg x_5 \vee x_3$$

$$\neg x_3$$

Clause set is unsatisfiable!

MaxSAT:

- Find a valuation which **maximizes** the number of satisfied clauses
- In the above set, a maximum of **10 clauses** can be satisfied simultaneously
- There are several variants of the MaxSAT problem

Variants of MaxSAT — 1

MaxSAT

- All clauses are *soft*, i.e. need not necessarily be satisfied.
- Maximize number of **satisfied soft** clauses
- Minimize number of **unsatisfied soft** clauses

Partial MaxSAT

- There are *hard* clauses, which *must* be **satisfied**
- Minimize number of **unsatisfied soft** clauses

Application example: car configuration

- Hard clauses: certain options/features must be present in car
- Soft clauses: additional options/features with dependencies
- Question: how many (and which) options can be ordered additionally for the car?

Variants of MaxSAT — 2

Weighted MaxSAT

- All clauses are soft
- All clauses carry *weights* in addition
- Minimize the sum of weights in **unsatisfied** clauses

Weighted Partial MaxSAT

- There are *hard* clauses, which must be **satisfied**
- There are soft clauses which carry weights
- Minimize the sum of weights in the **unsatisfied soft** clauses

Application example: car configuration

- Hard clauses: the configuration constraints given by the manufacturer (technical, legal, sales, ...)
- Soft clauses: options/features desired by the customer (typically unit clauses: a set of option codes desired to be all true), possibly with priorities or costs as weights
- Questions: optimal car configuration (sum of priorities maximized)

Notation

Notation: weighted clause

(c, w) : weighted clause

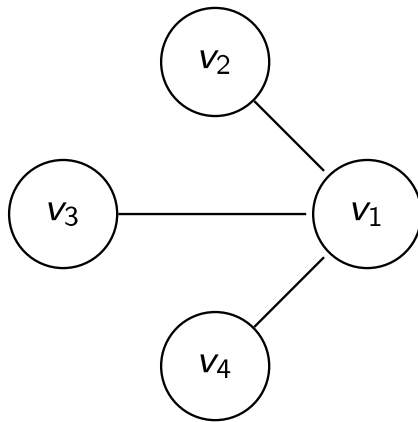
- c is a set of literals (a clause)
- w is a non-negative integer value or ∞ (or \top)
 - cost (penalty), if c is not satisfied

Notation: clause set

φ : set of weighted clauses

- **Soft clauses:** (c, w) with $w < \infty$
 - Cost, if c is not satisfied
- **Hard clauses:** (c, ∞)
 - Clause c must be satisfied

Modelling Example



Minimum Vertex Cover

- Vertex Cover $U \subseteq V$: For each edge $(v_i, v_j) \in E$ we have $v_i \in U$ or $v_j \in U$
- Minimum Vertex Cover: Vertex Cover U of minimal size

Example

Partial MaxSAT Encoding

- Variables: x_i for each vertex $v_i \in C$, with $x_i = 1$, iff $v_i \in U$
- **Hard clauses:** $(x_i \vee x_j)$ for each edge $(v_i, v_j) \in E$ (... is Vertex Cover)
- **Soft clauses:** $(\neg x_i)$ for each edge $v_i \in V$ (... is minimal)
 - the more variables are set to 0, the smaller the cover set

Encoding:

- $\varphi_H = \{(x_1 \vee x_2), (x_1 \vee x_3), (x_1 \vee x_4)\}$
- $\varphi_S = \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_4)\}$

MaxSAT vs. MinUNSAT

MinUNSAT

- Find valuation which minimizes the number U of unsatisfied clauses
 - this valuation also maximizes the number of satisfied clauses
- $\text{MinUNSAT}(\varphi) := U$.
- variants partial, weighted, and partial weighted are possible (like with MaxSAT)
 - Weighted MinUNSAT: minimize the weight of unsatisfied clauses

Relation

Let φ be a clause set, then:

$$|\varphi| = \text{MaxSAT}(\varphi) + \text{MinUNSAT}(\varphi)$$

Note: this relation holds for all variants

- Sometimes algorithms are given for MinUNSAT which can equally well be used for MaxSAT because of this relation

Example: MaxSAT vs. MinUNSAT

Example

- Let $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ with

$$\text{Hard} = \{\{x\}\}$$

$$\text{Soft} = \{(\{\neg x\}, 2), (\{y\}, 6), (\{\neg x, \neg y\}, 5)\}$$

- PartialWeightedMaxSAT(φ) = 6
Valuation: $x \mapsto 1, y \mapsto 1$
- PartialWeightedMinUNSAT(φ) = 7
Valuation: $x \mapsto 1, y \mapsto 1$
- We have:

$$13 = \text{PartialWeightedMaxSAT}(\varphi) + \text{PartialWeightedMinUNSAT}(\varphi)$$

Pseudo-Boolean Constraints & Optimization

Pseudo-Boolean Constraints

- Boolean variables: x_1, \dots, x_n
- Linear inequalities

$$\sum_{i \in N} a_i \ell_i \geq b, \quad \ell_i \in \{x_i, \bar{x}_i\}, x_i \in \{0, 1\}, a_i, b \in \mathbb{N}_0^+$$

Pseudo-Boolean Optimization

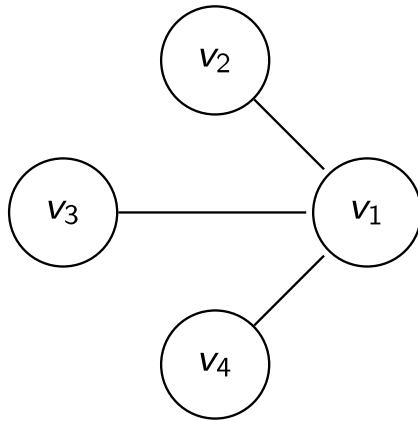
Minimize

$$\sum_{h \in N} w_h \cdot x_h$$

with respect to n PB constraints

$$\sum_{i \in N} a_{ij} \ell_i \geq b_j, \quad j \in \{1 \dots n\}, \ell_i \in \{x_i, \bar{x}_i\}, x_i \in \{0, 1\}, a_{ij}, b_j \in \mathbb{N}_0^+$$

Modelling Example



Minimum Vertex Cover

- Vertex Cover $U \subseteq V$: for each edge $(v_i, v_j) \in E$ we have $v_i \in U$ or $v_j \in U$
- Minimum Vertex Cover: Vertex Cover U of minimal size

Encoding as PBO

- Variables: x_i for each vertex $v_i \in V$, with $x_i = 1$ if $v_i \in U$
- PB Constraints: $x_i + x_j \geq 1$ for each edge $(v_i, v_j) \in E$
- Goal function: minimize number of variables with value= 1
 \Rightarrow i.e. minimize number of vertices in the Vertex Cover

Problem instance: $x_1 + x_2 \geq 1, x_1 + x_3 \geq 1, x_1 + x_4 \geq 1$

Goal function: $x_1 + x_2 + x_3 + x_4$

From (pure) MaxSAT to PBO

Starting from an unsatisfiable CNF formula φ :

- ① Generate φ' from φ :
 - Replace each clause c_i with $c'_i = c_i \cup \{r_i\}$
 - where r_i is a new variable (*selector variable, blocking variable*)
 - Now the problem can be trivially solved by setting all r_i to 1
- ② Minimize goal function $\sum r_i$

Example

- CNF formula φ :

$$\varphi = \{\{x_1, \neg x_2\}, \{x_1, x_2\}, \{\neg x_1\}\}$$

- Modified formula φ' :

$$\varphi = \{\{x_1, \neg x_2, r_1\}, \{x_1, x_2, r_2\}, \{\neg x_1, r_3\}\}$$

- PB Constraints: $x_1 + \bar{x}_2 + r_1 \geq 1, x_1 + x_2 + r_2 \geq 1, \bar{x}_1 + r_3 \geq 1$
- Minimize goal function: $r_1 + r_2 + r_3$

From Partial (Weighted) MaxSAT to PBO

Starting from a Partial (Weighted) MaxSAT instance with φ_H and φ_S

Generate PBO instance:

minimize $\sum w_i r_i$, such that φ_T holds, with

- $\varphi_T = \varphi'_H \cup \varphi'_S$
- Each hard clause (c, ∞) is mapped to a clause c in φ'_H
- Each soft clause (c, w) is mapped to a clause $(c_i \vee r_i)$ and the term $w_i r_i$ is added to the goal function

Example

- Original problem: $(\{x, y, \neg z\}, \infty), (\{x, \neg y\}, 4), (\{\neg x\}, 8), (\{x, z\}, 2)$
- $\varphi_T = \underbrace{(x, y, \neg z)}_{\varphi'_H}, \underbrace{(x, \neg y, r_1), (\neg x, r_2), (x, z, r_3)}_{\varphi'_S}$
- PB constraints: $x + y + \bar{z} \geq 1, x + \bar{y} + r_1 \geq 1, \bar{x} + r_2 \geq 1, x + z + r_3 \geq 1$
- Minimize goal function: $4r_1 + 8r_2 + 2r_3$

Application: Software Package Upgrades

- Available software packages: $P = \{p_1, \dots, p_n\}$
- Variable x_i for each package $p_i \in P$. $x_i = 1$, iff p_i is installed
- Constraints for each package p_i : (p_i, D_i, C_i)
 - D_i : Dependencies (required packages) when installing p_i
 - C_i : Conflicts (disallowed packages) when installing p_i
- Example problem: Maximum Installability
 - Maximal number of packages which can be installed
 - Package Constraints are hard clauses
 - Each individual package is a soft clause

Example

Package Constraints

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

MaxSAT Encoding

$$\begin{aligned}
 \varphi_H &= \{(\neg x_1 \vee x_2 \vee x_3), (\neg x_1 \vee \neg x_4), \\
 &\quad (\neg x_2 \vee x_3), (\neg x_2 \vee \neg x_4), (\neg x_3 \vee x_2), \\
 &\quad (\neg x_4 \vee x_2), (\neg x_4 \vee x_3)\} \\
 \varphi_S &= \{(x_1), (x_2), (x_3), (x_4)\}
 \end{aligned}$$

Cardinality Constraints

Question: How do we handle Cardinality Constraints?

- General form: $\sum_{j=1}^n x_j \bowtie k$ with $\bowtie \in \{<, \leq, =, \geq, >\}$
- In particular AtMost1 constraints: $\sum_{j=1}^n x_j \leq 1$

Solution 1

Use a special PB Solver

- Hard to keep up with progress in SAT Solving
- For SAT/UNSAT the best solvers already encode in CNF
 - such as Minisat+, QMaxSat, MSUnCore, (W)PM2

Solution 2

- encode cardinality constraints in CNF
- use a SAT solver

Equals, AtLeast1 & AtMost1 Constraints

Special treatment of constraints with right-hand side = 1 — occurring frequently:

- Car must have exactly one motor ...
- Exactly one graphics driver must be selected ...
- At most one SAT solver may be installed in Eclipse ...

Encodings

- $\sum_{j=1}^n x_j = 1$: encode as $(\sum_{j=1}^n x_j \leq 1) \wedge (\sum_{j=1}^n x_j \geq 1)$
- $\sum_{j=1}^n x_j \geq 1$: encode as $(x_1 \vee x_2 \vee \dots \vee x_n)$
- $\sum_{j=1}^n x_j \leq 1$: encode as:
 - Pairwise encoding:
 - clauses: $\mathcal{O}(n^2)$; no auxiliary variables
 - Sequential counter
 - clauses: $\mathcal{O}(n)$; auxiliary variables: $\mathcal{O}(n)$
 - Bitwise encoding:
 - clauses: $\mathcal{O}(n \log n)$; auxiliary variables $\mathcal{O}(\log n)$

General Cardinality Constraints

General form: $\sum_{j=1}^n x_j \leq k$ (or $\sum_{j=1}^n x_j \geq k$)

Encodings

- Sequential counter
 - clauses/variables: $\mathcal{O}(nk)$
- BDDs
 - clauses/variables: $\mathcal{O}(nk)$
- Sorting networks
 - clauses/variables: $\mathcal{O}(n \log^2 n)$
- Cardinality networks
 - clauses/variables: $\mathcal{O}(n \log^2 k)$

Pseudo-Boolean Constraints

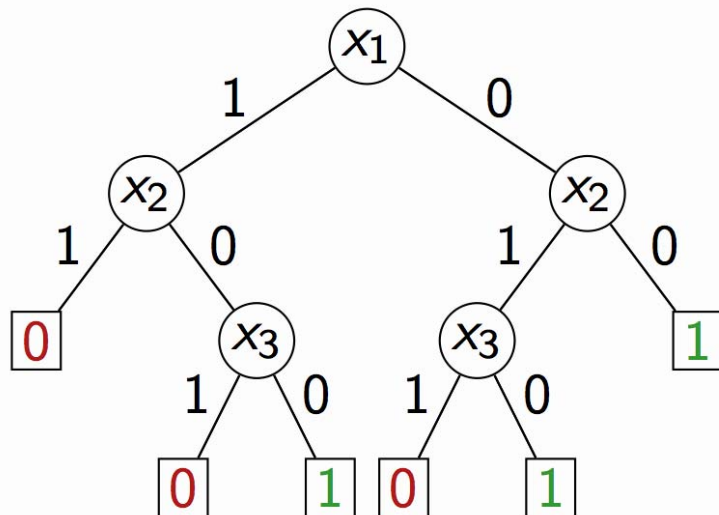
General form: $\sum_{j=1}^n a_j x_j \leq b$

- Encoding e.g. with BDDs (worst case: exponential number of clauses!)

Example

$$3x_1 + 3x_2 + x_3 \leq 3$$

- Encode a BDD, i.e. analyze variables by subtracting coefficients
- Convert BDD to CNF
- Simplify



Pseudo-Boolean Constraints

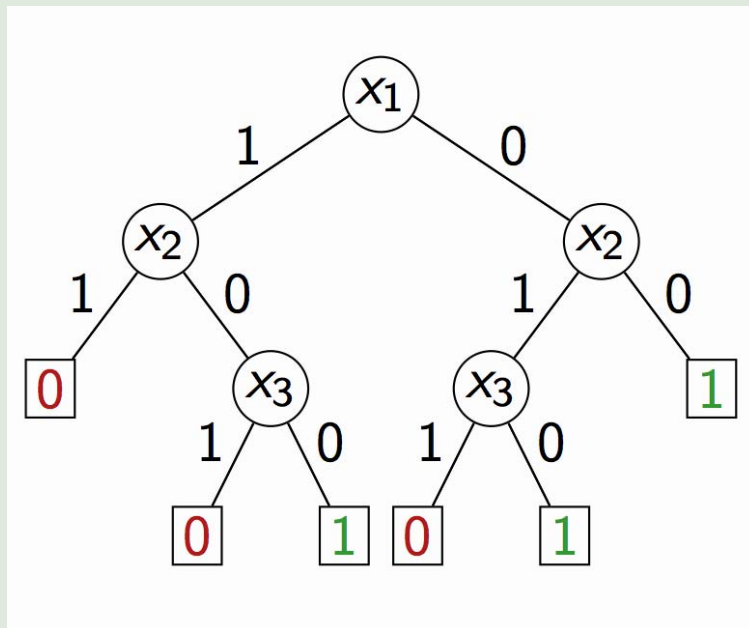
General form: $\sum_{j=1}^n a_j x_j \leq b$

- Encoding e.g. with BDDs (worst case: exponential number of clauses!)

Example

$$3x_1 + 3x_2 + x_3 \leq 3$$

- Encode a BDD, i.e. analyze variables by subtracting coefficients
- Convert BDD to CNF
- Simplify



- Conversion formula at node x :

$$F \equiv (\neg x \vee F|_{x=1}) \wedge (x \vee F|_{x=0})$$
- yields: $(\neg x_1 \vee \neg x_2) \wedge$
 $(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$
- simplifies to: $(\neg x_1 \vee \neg x_2) \wedge$
 $(\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_3)$

Some Practical Algorithms

- Branch & Bound
- Iterative (pure) SAT Solving
 - Step-wise approach from below (pure, simple, inefficient)
 - Leaping approach from above: LeBerge's Algorithm (for MaxSAT)
 - Binary Search (for MaxSAT)
- Iterative SAT Solving using the unsat core extension
 - Fu and Malik Algorithm for Partial MaxSAT
 - PM2 for Partial MaxSAT
 - WPM1 for Partial Weighted MaxSAT

MaxSAT by Iterative SAT Solving

Reducing MaxSAT Solving to SAT Solving

- Idea:
 - Iteratively call a SAT Solver, switching off clauses through blocking variables
- Advantage:
 - immediately use all modern efficient SAT Solving techniques (Unit propagation with Watched Literals, Clause Learning with non-chronological backtracking, etc) which do not (or only partially) apply to MaxSAT directly.
- Two approaches:
 - Reduction to (pure) SAT-Solving
 - Reduction to SAT-Solving with unsat core extension

Basic idea: step-wise approach from below

- Introduce a new blocking variable for each constraint: C_i becomes $C_i \vee b_i$.
- Introduce additional cardinality constraint on the blocking variables (blocked clauses): $\sum_{i=1}^m b_i \leq k$
- Iteratively solve $SAT((\bigcup C_i) \cup CNF(\sum_{i=1}^m b_i \leq k))$ für $k = 0, 1, \dots$ until enough (MinUNSAT many) clauses have been switched off (blocked)

Iterative MaxSAT: Leaping approach from above

The algorithm of Daniel LeBerre

Algorithm 1: LeBerre(φ)

Input: MaxSAT instance $\varphi = \{C_1, \dots, C_m\}$

$BV = \{b_1, \dots, b_m\}$

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$ // Add blocking variables

$ub \leftarrow m$

while SAT($\varphi \cup \text{CNF}(\sum_{i=1}^m b_i < ub)$) **do**

// Are blocking variables used at all?

if #satisfiedBlockingVariables > 0 **then**

└ $ub \leftarrow \text{\#satisfiedBlockingVariables}$

else

└ **return** 0

return ub

Literature: Biere *et al.* Handbook of Satisfiability, 19.6, S.625. IOS Press, 2009.