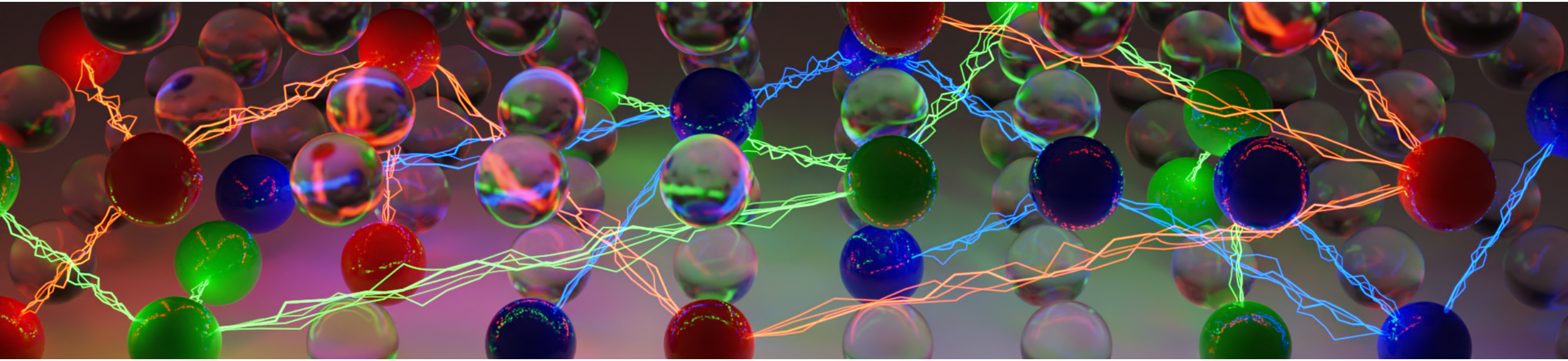
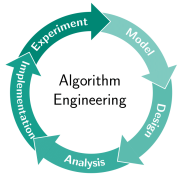


Practical SAT Solving



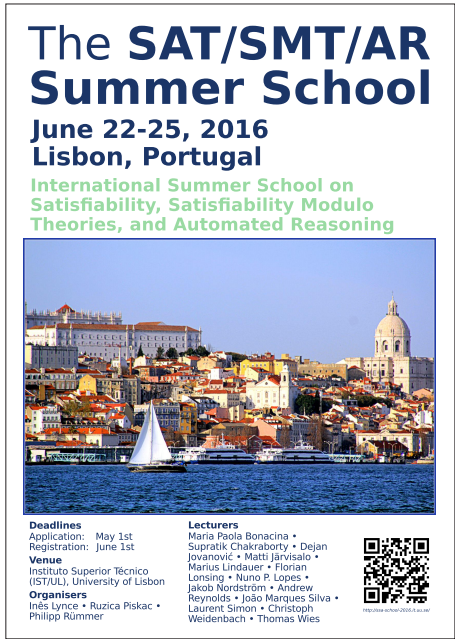
Lecture 13 – Maximum Satisfiability (MaxSAT)

Ashlin Iser, Dominik Schreiber | July 28, 2025

SAtRes
Scalable Automated Reasoning

Maximum Satisfiability

Today's lecture is based on the slides by Prof. Matti Järvisalo presented at 2016 SAT Summer School.



http://ssa-school-2016.it.uu.se/programme/#maxSAT

Maximum Satisfiability (MaxSAT)

Exact Boolean Optimization Paradigm

- Basic concepts: MaxSAT, complexity, and applications
- Overview of algorithmic approaches to MaxSAT
  - Branch and Bound
  - Integer Programming (IP)
  - Linear SAT-UNSAT (LSU) Approach
  - Core-guided Approach
  - Implicit Hitting Sets (IHS)

Boolean Optimization

Motivation

Most real-world problems involve an optimization component. There is a high demand for automated approaches to finding good solutions to computationally hard optimization problems.

Examples

- Find a shortest path/plan/execution to a goal/error state: *Planning, model checking, debugging, . . .*
- Find a smallest explanation: *Explainable machine learning, . . .*
- Find a least resource-consuming schedule: *Scheduling, logistics, . . .*
- Find an optimal swap cycle in a kidney exchange: *Healthcare*

Benefits

- Resource savings (time, workforce, energy, material, . . .)
- Life savings (e.g., in healthcare, disaster response, . . .)
- Accuracy and proofs of optimality
- Better approximations by optimally solving simplified problem representations

MaxSAT

Classic Definition and Terminology

Input: CNF formula F (set of clauses)
Task: Find an assignment τ that maximizes the number of satisfied clauses

Central Generalizations

- **Weighted MaxSAT:** Each clause C has a weight w_C, and the goal is to maximize the total weight of satisfied clauses.
- **Partial MaxSAT:** Some clauses are hard (infinite weight); only soft clauses can be violated.
- **Weighted Partial MaxSAT:** Mix of hard clauses and weighted soft clauses.

Terminology

- **Solution:** Assignment satisfying all hard clauses
- **Cost:** Sum of weights of falsified soft clauses
- **Optimal Solution:** One that minimizes the cost

Generic Linear Optimization Paradigms

Relationship with Generic Optimization: Each MaxSAT instance can be reencoded such that all soft clauses are unit clauses. Soft unit clauses can then be interpreted as variables in an objective function of a generic optimization problem.

Given a conjunction of constraints $\bigwedge_k C_k$ of the form $C_k := \sum_{i=1}^{n} c_{ki} x_i \leq b_k$ (with coefficients c_{ki} and bound b_k),
find an assignment to the variables x_i that satisfies all constraints
and that maximizes the objective function $\sum_{i=1}^{n} c_i x_i$ (with coefficients c_i).

Constrained Optimization Paradigms

- Maximum Satisfiability (MaxSAT)
  - Variables x are Boolean, Coefficients $c \in \{-1, 0, 1\}$, Bound $b = -1$
- Pseudo-Boolean Optimization (PBO)
  - Variables x are Boolean, Coefficients c, and Bound b are Integers
- Integer-Linear Programming (ILP)
  - Variables x, Coefficients c, and Bounds b are Integers

MaxSAT Applications

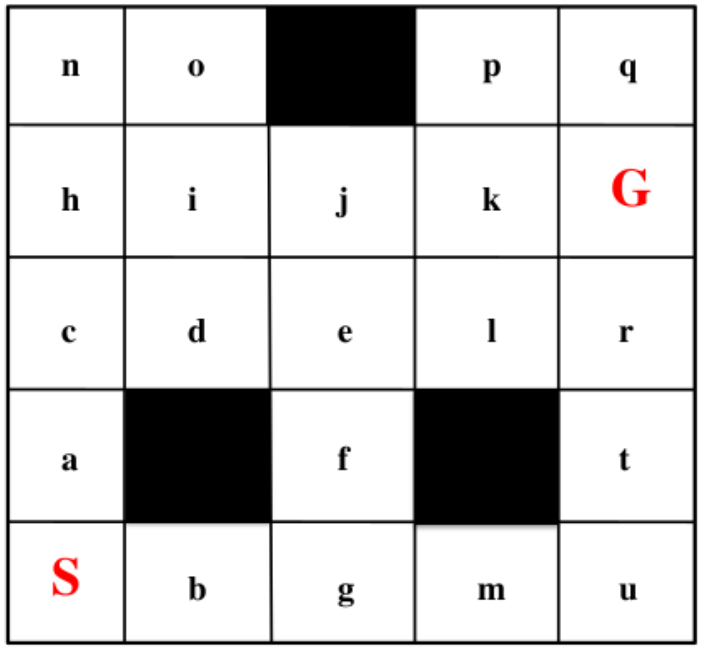
MaxSAT solvers are particularly successful on inherently Boolean problems.

- **Hardware Design**: Placement, Routing, Debugging, Verification
- **Planning, Scheduling, Resource Allocation**
- **Product Configuration**, **Dependency Management**, **Software Package Management**
- **Causal Reasoning, Argumentation, Formal XAI**
- ...

Toy Example

Encoding Shortest Paths

- Grid-based shortest path problem from S to G
- Horizontal/vertical moves only; blocked cells not allowed
- Not a practical MaxSAT application, but useful for illustration



Toy Example

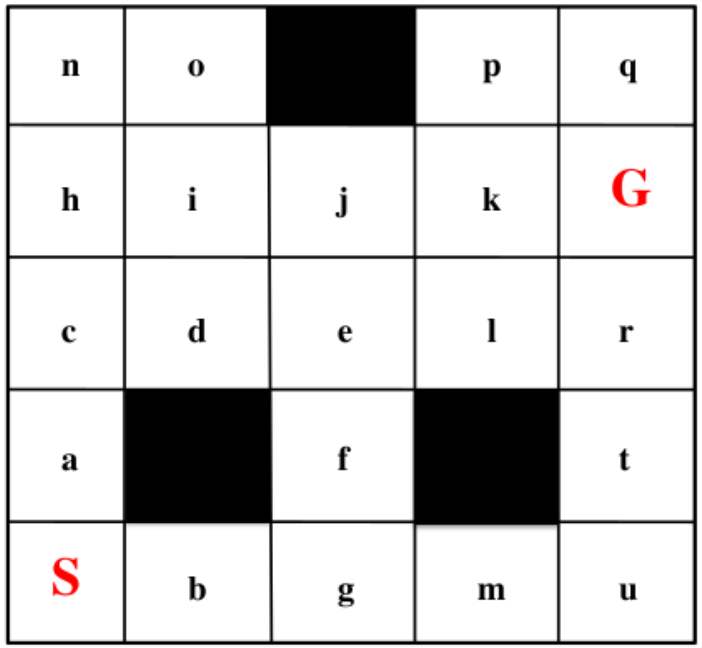
Encoding Shortest Paths

Basic Encoding Idea: One Boolean variable per unblocked square

- *S*, *G* must be visited (hard unit clauses)
- All other squares: “Prefer not to visit” (soft unit clauses with weight 1)

MaxSAT minimizes the number of visited squares.

Without further constraints that formulation only visits *S* and *G*.



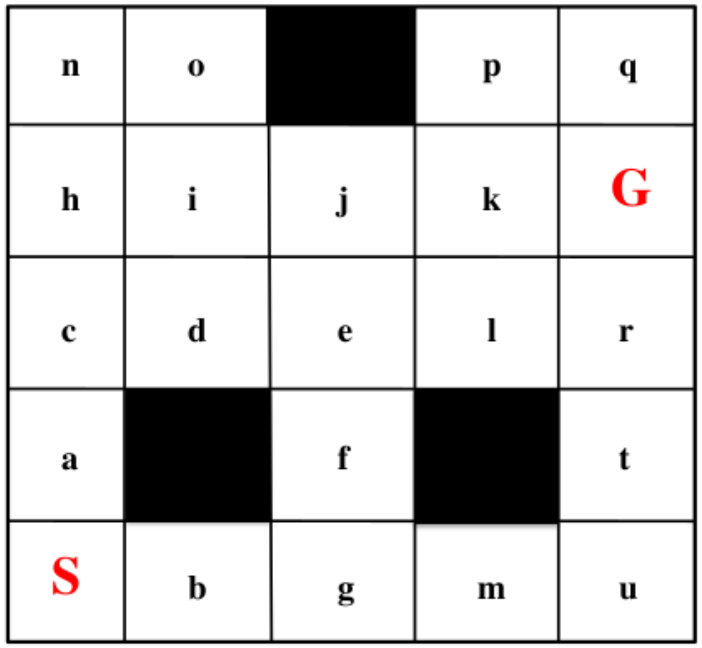
Toy Example

Encoding Shortest Paths

Ensure a valid path between *S* and *G*.

Constraint 1: *S* and *G* must have exactly one visited neighbor

Constraint 2: All other visited squares must have exactly two visited neighbors



Toy Example

Encoding Shortest Paths

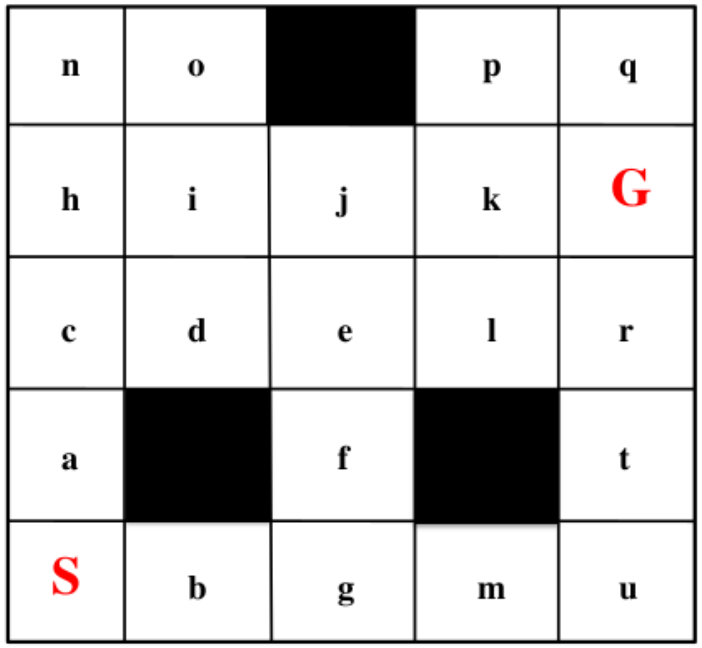
Ensure a valid path between S and G.

Constraint 1: S and G must have exactly one visited neighbor

- For S: $a + b = 1$ CNF: $\{a, b\}, \{\neg a, \neg b\}$
- For G: $k + q + r = 1$ CNF: $\{k, q, r\}, \{\neg k, \neg q\}, \{\neg k, \neg r\}, \{\neg q, \neg r\}$

Constraint 2: All other visited squares must have exactly two visited neighbors

- Example: for square e, if e is visited, then $d + j + l + f = 2$
- Requires encoding a cardinality constraint in CNF

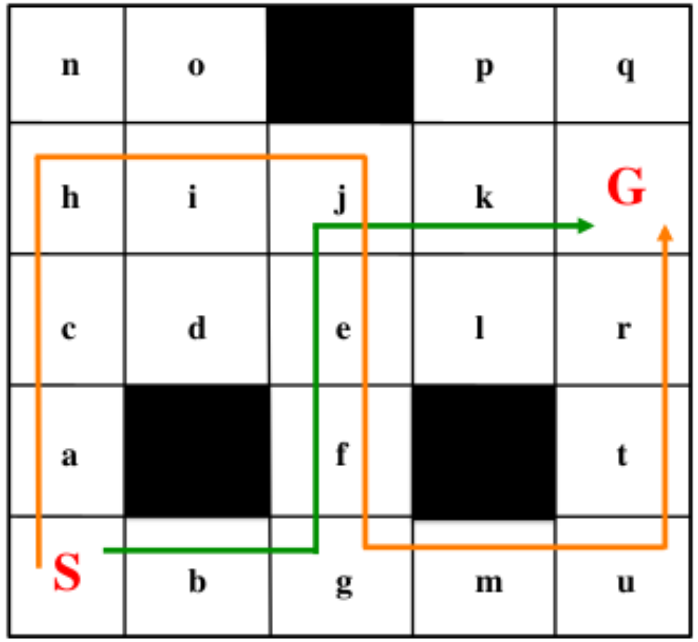


Toy Example

Encoding Shortest Paths

Path Properties: Every solution to the hard clauses defines a valid path from S to G.

- Each visited square falsifies a soft clause (e.g., $\neg x$)
- MaxSAT solution is a shortest path (minimum number of visited squares)



Example: Two valid paths from S to G.

- Orange path: 14 visited squares
- Green path: 8 visited squares (optimal)

Representing High-Level Soft Constraints

MaxSAT can represent high-level soft constraints compactly.

Softening an $\mathcal{NP}$-Constraint

- Let $\mathcal{C}$ be a finite-domain soft constraint with weight $W_{\mathcal{C}}$
- Encode $\mathcal{C}$ into CNF: $CNF(\mathcal{C}) = C_1 \wedge C_2 \wedge \cdots \wedge C_m$
- Introduce fresh variable a, add hard clauses: $(C_i \vee a)$ for all i
- Add soft clause: $(\neg a)$ with weight $W_{\mathcal{C}}$

A Glimpse into the Polynomial Hierarchy

Defining complexity beyond $\mathcal{NP}$

Definition: Polynomial Hierarchy (without co-$\mathcal{NP}$)

- $\Sigma_0^p := \Delta_0^p := \mathcal{P}$
- $\Sigma_{k+1}^p = \mathcal{NP}^{\Sigma_k^p}$ (nondeterministic poly-time *with* a Σ_k^p oracle)
- $\Delta_{k+1}^p = \mathcal{P}^{\Sigma_k^p}$ (deterministic poly-time *with* a Σ_k^p oracle)
- $\mathcal{P} \subseteq \mathcal{NP} \subseteq \Delta_2^p \subseteq \Sigma_2^p \subseteq \ldots \subseteq \mathrm{PH} \subseteq \mathrm{PSPACE}$

- Level 1: $\Sigma_1^p = \mathcal{NP}, \quad \Delta_1^p = \mathcal{P}$
- Level 2: $\Sigma_2^p = \mathcal{NP}^{\mathcal{NP}}, \quad \Delta_2^p = \mathcal{P}^{\mathcal{NP}}$
- $\Delta_2^p \longleftarrow$ deciding things is as easy as polynomially many calls to an $\mathcal{NP}$ oracle (MaxSAT)

Note that the slides leave out co-$\mathcal{NP}$ to simplify the depiction: $\Pi_0^p = \mathcal{P}$, $\Pi_1^p =$ co-$\mathcal{NP}$, $\Pi_2^p =$ co-$\mathcal{NP}^{\mathcal{NP}}$, ...

Practical MaxSAT Solving

Input Format, Solvers

Standard Solver Input Format: DIMACS WCNF

- Like DIMACS CNF: Variables indexed from 1 to n, Negation: $-i$ means $\neg x_i$, Clauses terminated with 0
- Header line: `p wcnf <#vars> <#clauses> <top>`
- Clause weight is first integer in line; if weight $\geq$ `top` $\rightarrow$ hard clause

Push-Button Solvers / Black-box Solvers

- Input: in standard WCNF format
- Output: provably optimal solution or `UNSATISFIABLE`
- Internally rely on CDCL SAT solvers to prove unsatisfiability of subsets of clauses
- Find recent MaxSAT solvers and benchmarks at `https://maxsat-evaluations.github.io/2026/`

Recap.

So far: MaxSAT

- Powerful paradigm for Boolean optimization
- Wide range of real-world problems
- Complexity: Δ_2^p-complete
- Standard input format: DIMACS WCNF
- Push-button solvers are available and effective

Next up

Fundamental algorithms for solving MaxSAT

Algorithms for MaxSAT Solving

Example Implementations / Historic Solvers

- **Branch and Bound:** MaxSatz, ahmaxsat
- **Direct Integer Programming:** IP Encoding + IP Solver (e.g., CPLEX, Gurobi)
- **Iterative, Linear SAT to UNSAT (LSU):** QMaxSAT
- **Core-Guided:** Eva, MSCG, OpenWBO, WPM, maxino
- **IP-SAT Hybrids:** MaxHS, LMHS

Branch and Bound

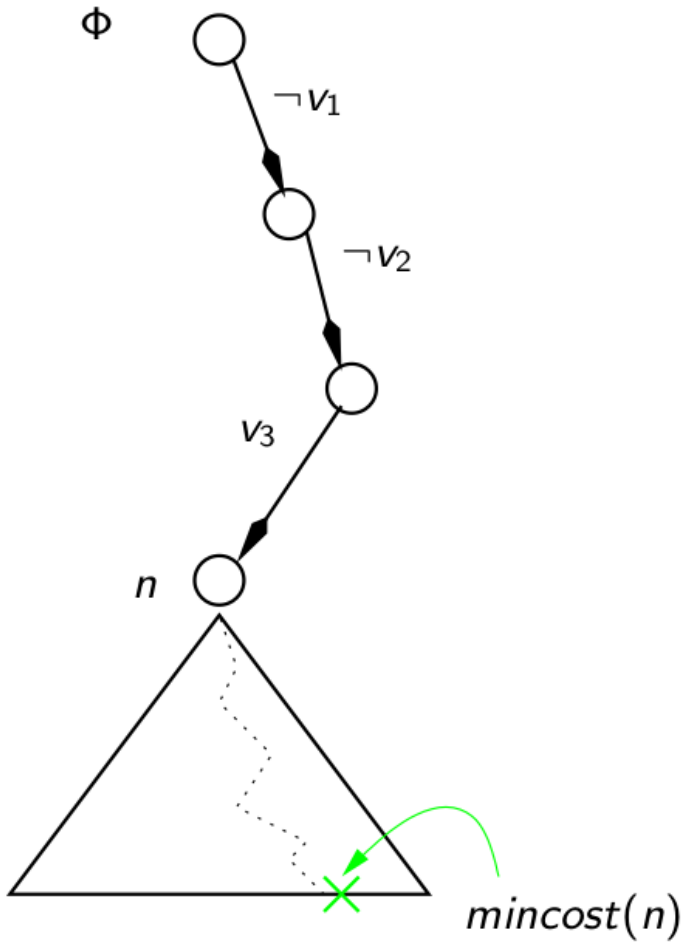
Classic method for optimization over search trees

Effective on small, combinatorially hard problems (e.g., Max-Clique),
but scalability issues with thousands of variables

- UB = Maintain upper bound (UB) on current best solution cost
- mincost(n) = minimum cost achievable under node n
- Backtrack if $mincost(n) \geq UB$
  $\rightarrow$ no solution under node n can improve the current best solution UB

Basic technique:

- Compute lower bound (LB) such that mincost(n) $\geq$ LB
- If LB $\geq$ UB, then backtrack ($\Rightarrow$ mincost(n) $\geq$ LB $\geq$ UB)



Branch and Bound

Lower Bounds by UNSAT Cores

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

- Strategy: find UNSAT cores, i.e. unsatisfiable subsets of clauses
  $\longrightarrow$ At least one clause in an UNSAT core has to be falsified
- Example: $\kappa = \{(\{x\}, 2), (\{\neg x\}, 3)\}$ is UNSAT core
  - replace it with $\kappa' = \{(\emptyset, 2), (\{\neg x\}, 1)\}$
  - Cost of $\emptyset$ (=minimum cost) gets increased by 2
    $\longrightarrow$ 2 is a lower bound
  - The cost of each truth assignment is preserved
- Repeat:
  1. Detect unsatisfiable core κ
  2. Apply sound transformation to increase cost($\emptyset$)
  3. Stop if no further LB improvement possible or LB $\geq$ UB

MaxSAT by Integer Programming (IP)

Using IP solvers as MaxSAT engines

- IP solvers are widely used in Operations Research (IBM CPLEX, Gurobi, SCIP, etc.)
- Solve problems with linear constraints and integer variables
- Very effective on many standard optimization problems

  Do not dominate native MaxSAT solvers on “very Boolean” problems

MaxSAT Encoding into IP

1. Relax each soft clause C_i using a new variable r_i
2. Convert each clause to linear constraint:

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

3. Boolean variables become 0-1 bounded integers
4. Objective function:

$$\min \sum_{C_i \in F_s} w_i \cdot r_i$$

SAT-Based MaxSAT Solving

The most widely used modern approach

- Solve a sequence of SAT instances that ask for different values of k:
  Is there a truth assignment falsifying at most k soft clauses?
- SAT-based MaxSAT algorithms mainly do two things:
  1. Develop better ways to encode this decision problem.Algorithms for solving MaxSAT
  2. Find ways to exploit information obtained from the SAT solver at each stage in the next stage.

Assume unit-weight soft clauses for now . . .

Methods for SAT-Based MaxSAT

- **Iterative Search:** Iteratively increase (decrease) k until SAT (UNSAT)
- **Core-Based Methods:** Use unsatisfiable cores to guide search
- **Hybrid Methods:** Combine SAT solving with integer programming

Iterative Search

To check whether F has a solution of cost $\leq k$, solve: $(C_1 \vee r_1) \wedge \cdots \wedge (C_n \vee r_n) \wedge \left(\sum_{i=1}^{n} r_i \leq k\right)$, and iterate over $k = 1, 2, \ldots$ until optimal k is found.

Iterating over k

- **Linear Search (UNSAT to SAT):** (not efficient)
  Start at $k = 1$, increment until SAT
- **Linear Search (SAT to UNSAT):** (can be effective)
  - Find model π for hard clauses, let $k = \#\text{violated soft clauses} - 1$
  - Try solving again with lower k until UNSAT
  - If SAT: set k to $\#$violated soft clauses and repeat
  - If UNSAT: last SAT solution is optimal
- **Binary Search:** (can be effective with core-based reasoning)
  - Initialize: $LB = 0$, $UB = \#\text{soft clauses}$
  - Check $k = \lfloor \frac{LB+UB}{2} \rfloor$
  - If SAT: $UB = k$, else $LB = k + 1$
  - Stop when $UB = LB + 1$, then UB is optimal.

SAT-Based MaxSAT Solving using UNSAT Cores

Motivation

Adding linear cardinality constraints over all soft clauses is too loose:

- One relaxation variable r_i per soft clause, could be well over 100k of variables
- Linear cardinality constraints over all soft clauses are too loose:
  no information about which relaxation variables to assign to 1
- SAT solver must explore many subsets of soft clauses

Unsatisfiable Cores in MaxSAT

Core-based approach gives more powerful constraint over which particular soft clauses to relax.

- **UNSAT Core:** A subset $F'_s \subseteq F_s$ s.t. $F_h \wedge F'_s$ is UNSAT
- At least one clause in each core must be falsified
- Instead of iteratively ruling out non-optimal solutions, iteratively find and rule out UNSAT cores
- Typically cores are much smaller than full soft clause set

Core-Guided MaxSAT Algorithms: Fu-Malik

The Original

- First core-guided MaxSAT algorithm [Fu & Malik, 2006]
- Iterative approach:
  1. Find an UNSAT core
  2. Relax clauses in the core with new variables
  3. Add an AtMost-1 constraint over new relaxation vars
- Repeat until the formula becomes SAT
- Each iteration lowers the cost of solutions by 1 (in the unweighted case)

Fu-Malik: Example

- Initial Formula:
  $C_1 = x_6 \vee x_2$, $C_2 = \neg x_6 \vee x_2$, $C_3 = \neg x_2 \vee x_1$, $C_4 = \neg x_1$, $C_5 = \neg x_6 \vee x_8$,
  $C_6 = x_6 \vee \neg x_8$, $C_7 = x_2 \vee x_4$, $C_8 = \neg x_4 \vee x_5$, $C_9 = x_7 \vee x_5$, $C_{10} = \neg x_7 \vee x_5$,
  $C_{11} = \neg x_5 \vee x_3$, $C_{12} = \neg x_3$
- Core 1: $\{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}$
- Add relaxation variables r_1 to r_6, and AMO constraint $\sum r_i \leq 1$
- Core 2: $\{C_1, C_2, C_3, C_4, C_9, C_{10}, C_{11}, C_{12}\}$
- Add relaxation variables r_7 to r_{14} to these clauses, and AMO constraint $\sum_{i=7}^{14} r_i \leq 1$
- Now the instance is SAT, and the optimal cost is the number of iterations (2 here)
- Final Formula:
  $C_1 = x_6 \vee x_2 \vee r_7$, $C_2 = \neg x_6 \vee x_2 \vee r_8$, $C_3 = \neg x_2 \vee x_1 \vee r_1 \vee r_9$, $C_4 = \neg x_1 \vee r_2 \vee r_{10}$, $C_5 = \neg x_6 \vee x_8$
  $C_6 = x_6 \vee \neg x_8$, $C_7 = x_2 \vee x_4 \vee r_3$, $C_8 = \neg x_4 \vee x_5 \vee r_4$, $C_9 = x_7 \vee x_5 \vee r_{11}$, $C_{10} = \neg x_7 \vee x_5 \vee r_{12}$
  $C_{11} = \neg x_5 \vee x_3 \vee r_5 \vee r_{13}$, $C_{12} = \neg x_3 \vee r_6 \vee r_{14}$, $\sum_{i=1}^{6} r_i \leq 1$, $\sum_{i=7}^{14} r_i \leq 1$

Other Core-Guided MaxSAT Algorithms

MSU3 Algorithm by Marques-Silva and Planes (2007)

Differences to Fu-Malik:

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

OpenWBO Algorithm by Martins, Joshi, Manquinho, and Lynce, 2014

Combines MSU3 with incremental construction of the cardinality constraint:
$\rightarrow$ Each new constraint builds on the encoding of the previous constraint

WPM2 Algorithm by Ansótegui, Bonet, and Levy, 2013a

Proposes a method for dealing with overlapping cores: groups intersecting cores into disjoint covers.
The cores might not be disjoint but the covers will be
$\rightarrow$ at-most-k constraints over the soft clauses in a cover
$\rightarrow$ at-least-k constraint over the clauses in a core

Implicit Hitting Set Algorithms for MaxSAT

Combining Integer Programming with SAT solving

Hitting Sets

Given a collection of sets $\mathcal{S}$ of elements, a **hitting set** H is a subset of elements that intersects all sets $S \in \mathcal{S}$.
A hitting set H is optimal if no smaller hitting set exists.

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

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

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

Implicit Hitting Set Algorithms for MaxSAT

Hitting Set Problem as Integer Programming

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

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

MaxSAT Solving with Implicit Hitting Sets

Iterate over the following steps:

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

. . . until the SAT solver returns a satisfying assignment.

Implicit Hitting Set Algorithms for MaxSAT

Optimizations

Optimizations

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

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

IHS MaxSAT Solvers

For more on some of the details, see

- **MaxHS** [Davies and Bacchus, 2011 and 2013]
- **LMHS** [Saikko, Berg, and Järvisalo, 2016]

IHS Algorithms for MaxSAT are among the best performing solvers today, and work well on a wide range of problems, particularly on large instances with many different weights on soft clauses.