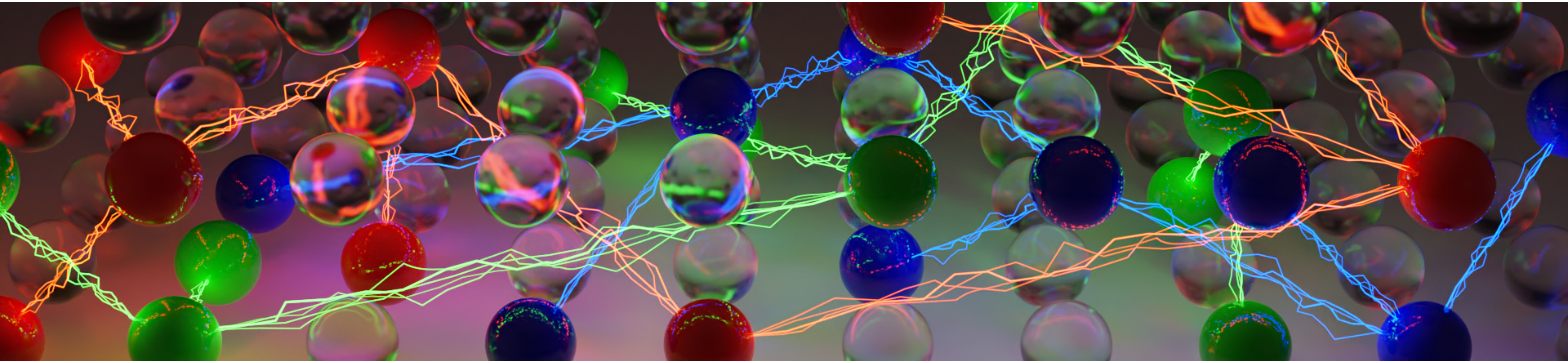


Practical SAT Solving

Lecture 13 – Maximum Satisfiability (MaxSAT)

Markus Iser, Dominik Schreiber | July 28, 2025

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, . . .*

Benefits of provably optimal solutions

- Resource savings: Time, Workforce, Energy, Material, . . .
- Accuracy
- Better approximations by optimally solving simplified problem representations

Key Challenge: Scalability of exactly solving instances of NP-hard optimization problems

Generic Linear Optimization Paradigms

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

Constrained Optimization Paradigms

- Integer-Linear Programming (ILP)
  - Variables x_i, Coefficients c_i, d_i, and Bounds b are Integers
  - Algorithms: e.g. Branch-and-Cut with Simplex
- Pseudo-Boolean Optimization (PBO)
  - Variables x_i are Boolean, Coefficients c_i, d_i, and Bound b are Integers
  - Algorithms: e.g. CDCL-based
- Maximum Satisfiability (MaxSAT)
  - Variables x_i are Boolean, Coefficients $c_i, d_i \in \{-1, 0, 1\}$, Bound $b = -1$
  - Algorithms: e.g. CDCL-based

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); soft clauses can be violated.
- **Weighted Partial MaxSAT:** Mix of hard clauses and weighted soft clauses.

Relationship with Generic Optimization: Each of these variants can be reencoded such that all soft clauses are unit clauses. Soft unit clauses can then be interpreted as variables in the objective function.

Terminology

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

MaxSAT Applications

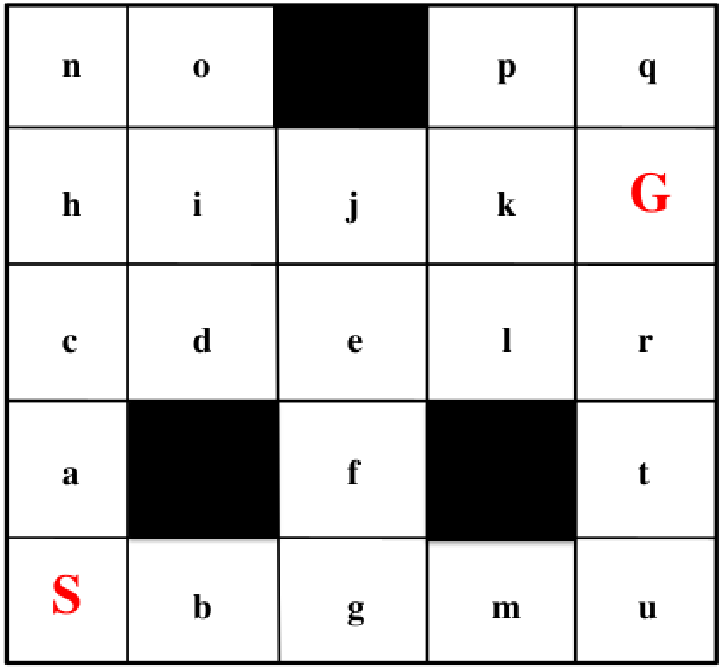
MaxSAT solvers are particularly successful on inherently Boolean problems.

- Placement/Routing/Debugging/Verification in Hardware Design
- Planning, Scheduling, Resource Allocation
- Product Configuration
- Software Package Management
- Causal Discovery, Argumentation, Formal XAI
- Max-Clique
- . . . and many more!

Central to success: Advances in MaxSAT solver technology.

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



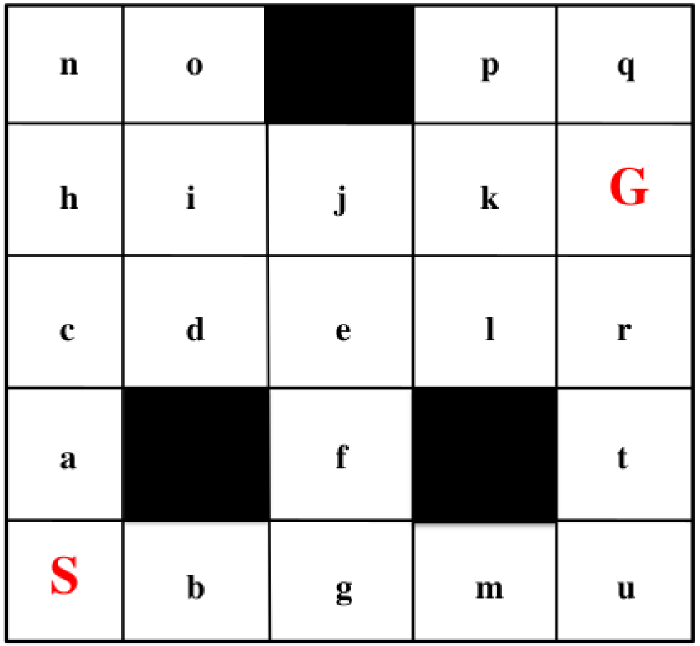
Example: Encoding Shortest Paths

Basic Encoding Idea:

- One Boolean variable per unblocked square
- S, G must be visited (hard unit clauses)
- All other squares: soft unit clauses (e.g., $\neg a$) with weight 1 ("Prefer not to visit")

MaxSAT minimizes the number of visited squares.

Without further constraints that formulation only visits S and G.



Example: Encoding Shortest Paths

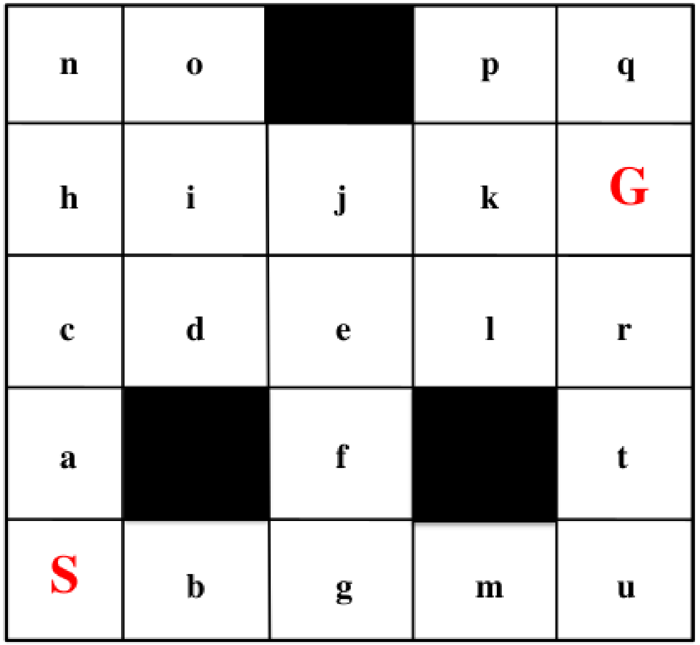
Ensure a valid path between S and G.

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

- For S:
  $a + b = 1$
  CNF: $(a \vee b), (\neg a \vee \neg b)$
- For G:
  $k + q + r = 1$
  CNF: $(k \vee q \vee r), (\neg k \vee \neg q), (\neg k \vee \neg r), (\neg q \vee \neg r)$

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

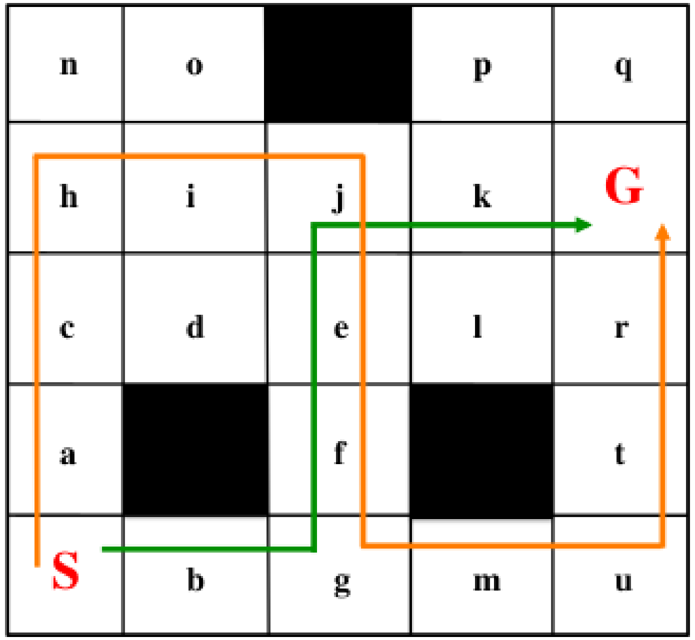
- Example: for square e, if e is visited, then $d + j + l + f = 2$
- Requires encoding a cardinality constraint in CNF



Example: 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)



- 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 $W_{\mathcal{C}}$

MaxSAT: Complexity

- **Decision version:** $\mathcal{NP}$-complete
  - Given CNF F, integer k: is there an assignment satisfying at least k clauses?
- **Optimization version:** $\text{FP}^{\mathcal{NP}}$-complete
  - Solvable with a polynomial number of calls to an $\mathcal{NP}$ oracle
  - SAT solver acts as the $\mathcal{NP}$ oracle in practice
  - Same as TSP: polynomial-time computation using an $\mathcal{NP}$ oracle
- **Hard to approximate:** APX-complete
  - Constant-factor approximation possible
  - No poly-time approximation scheme (PTAS) unless $\mathcal{P} = \mathcal{NP}$

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
- Examples: Open-source MaxSAT Solvers
  - OpenWBO — http://sat.inesc-id.pt/open-wbo/
  - MaxHS — http://maxhs.org
  - LMHS — http://www.cs.helsinki.fi/group/coreo/lmhs/

Recap.

So far

- MaxSAT is a powerful paradigm for Boolean optimization
- Can be used to model and solve a wide range of real-world problems
- Complexity: $\mathrm{FP}^{\mathcal{NP}}$-complete
- Standard input format: DIMACS WCNF
- Push-button solvers are widely available and effective

Next up

Algorithms for solving MaxSAT

Algorithms for MaxSAT Solving

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

Branch and Bound

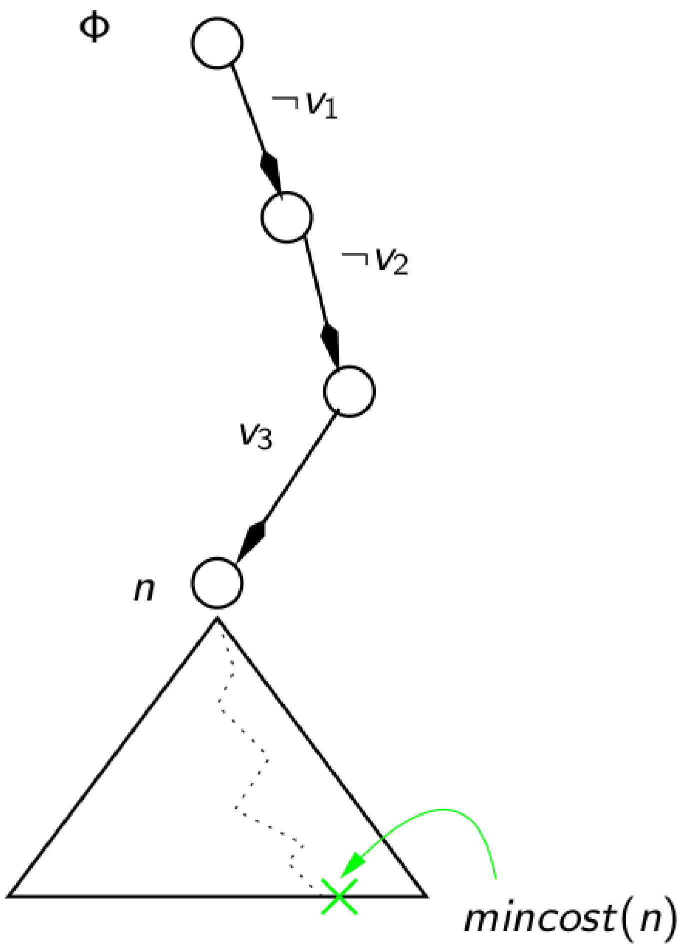
Classic method for optimization over search trees

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 Cores

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

- Strategy: find unsatisfiable sets of clauses (UNSAT cores)
- Each core forces at least one clause to be falsified
- Example:
  - $\kappa = \{(x, 2), (\neg x, 3)\}$ is unsatisfiable; replace with $\kappa' = \{(\emptyset, 2), (\neg x, 1)\}$
  - $\{(x, 2), (\neg x, 3)\} \rightarrow \{(\emptyset, 2), (\neg x, 1)\}$
  - Cost of $\emptyset$ increased by 2
    $\Rightarrow$ 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 widely used in Operations Research, e.g. IBM CPLEX, Gurobi, SCIP, etc.
- Solve problems with linear constraints and integer variables
- Very effective on many standard optimization problems

  But 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.
  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 k until SAT
- **Core-Based Methods:** Use unsatisfiable cores to guide search
- **Hybrid Methods:** Combine SAT solving with integer programming

Iterative Search

Basic Approach

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

Iterating over k

- **Linear Search:** (not efficient)
  Start at $k = 1$, increment until SAT
- **Binary Search:** (effective with core-based reasoning)
  - Initialize: $LB = 0$, $UB = \#$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.
- **Linear Search (SAT to UNSAT):** (can be effective)
  - Find model π for hard clauses, let $k = \#$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

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

- 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:
  $C_1 = x_6 \vee x_2,\quad C_2 = \neg x_6 \vee x_2,\quad C_3 = \neg x_2 \vee x_1,\quad C_4 = \neg x_1,\quad C_5 = \neg x_6 \vee x_8,$
  $C_6 = x_6 \vee \neg x_8,\quad C_7 = x_2 \vee x_4,\quad C_8 = \neg x_4 \vee x_5,\quad C_9 = x_7 \vee x_5,\quad C_{10} = \neg x_7 \vee x_5,$
  $C_{11} = \neg x_5 \vee x_3,\quad C_{12} = \neg x_3$
- 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:
  $C_1 = x_6 \vee x_2 \vee r_7,\quad C_2 = \neg x_6 \vee x_2 \vee r_8,\quad C_3 = \neg x_2 \vee x_1 \vee r_1 \vee r_9,\quad C_4 = \neg x_1 \vee r_2 \vee r_{10},\quad C_5 = \neg x_6 \vee x_8$
  $C_6 = x_6 \vee \neg x_8,\quad C_7 = x_2 \vee x_4 \vee r_3,\quad C_8 = \neg x_4 \vee x_5 \vee r_4,\quad C_9 = x_7 \vee x_5 \vee r_{11},\quad C_{10} = \neg x_7 \vee x_5 \vee r_{12}$
  $C_{11} = \neg x_5 \vee x_3 \vee r_5 \vee r_{13},\quad C_{12} = \neg x_3 \vee r_6 \vee r_{14},\quad \sum_{i=1}^{6} r_i \leq 1,\quad \sum_{i=7}^{14} r_i \leq 1$

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
- Instead of adding one AtMost-1/Exactly-1 constraint per iteration:
  Update the AtMost-k, k noting the k-th iteration

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.

Recap.

Today's Lecture

- MaxSAT is a powerful paradigm for Boolean optimization
- Can be used to model and solve a wide range of real-world problems
- Complexity: $\mathrm{FP}^{\mathcal{NP}}$
- Standard input format: DIMACS WCNF
- Push-button solvers are widely available and effective
- Several algorithmic approaches to MaxSAT solving
- Core-guided MaxSAT solving is a powerful approach
- Implicit Hitting Set algorithms are among the best performing MaxSAT solvers