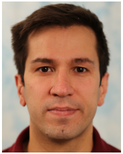
# Advances in Maximum Satisfiability

Jeremias Berg[1] Matti Järvisalo[1] Ruben Martins[2]

[1]University of Helsinki
Finland

[2]Carnegie Mellon University
USA

September 4, 2020 ECAI'20 Online

# What This Tutorial is About

## Maximum Satisfiability—MaxSAT

Exact Boolean optimization paradigm

- Builds on the success story of Boolean satisfiability (SAT) solving
- Great recent improvements in practical solver technology
- Expanding range of real-world applications

Offers an alternative to e.g. integer programming

- Solvers provide provably optimal solutions
- Propositional logic as the underlying declarative language: especially suited for inherently "Boolean" optimization problems

# Tutorial Outline

Three parts:

1. Motivation and basic concepts
2. Practical algorithms for MaxSAT
3. Applications and encodings

# Success of SAT

### The Boolean satisfiability (SAT) Problem

Input: A propositional logic formula $F$.
Task: Is $F$ *satisfiable*?

# Success of SAT

### The Boolean satisfiability (SAT) Problem

Input: A propositional logic formula $F$.
Task: Is $F$ *satisfiable*?

### SAT is a Great Success Story

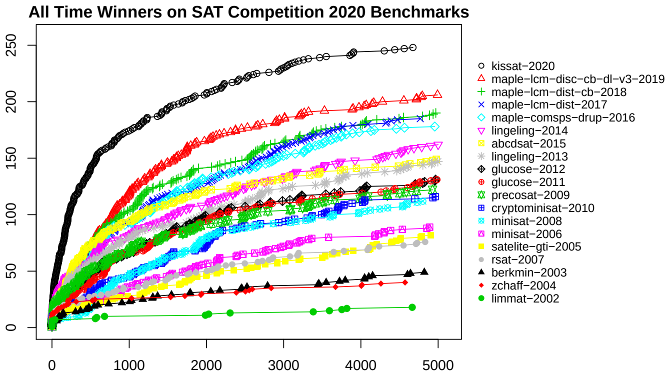
Not merely a central problem in *theory*:

Remarkable improvements since mid 90s in **SAT solvers**:
*practical decision procedures for SAT*

- *Find solutions* if they exist
- *Prove* non-existence of solutions

# SAT Solvers

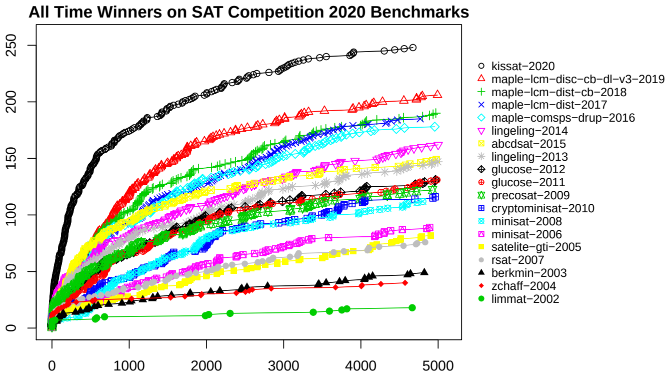
*From 100 variables, 200 constraints (early 90s)*
*up to >10,000,000 vars. and >50,000,000 clauses. in 20 years.*




Plot provided by Armin Biere

# SAT Solvers

*From 100 variables, 200 constraints (early 90s)*
*up to >10,000,000 vars. and >50,000,000 clauses. in 20 years.*




Plot provided by Armin Biere

Core NP search procedures for solving various types of computational problems

# Optimization

Most real-world problems involve an optimization component

Examples:

- Find a **shortest** path/plan/execution/...to a goal state
  - Planning, model checking, ...
- Find a **smallest** explanation
  - Debugging, configuration, ...
- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...
- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

# Optimization

Most real-world problems involve an optimization component

Examples:

- Find a **shortest** path/plan/execution/...to a goal state
  - Planning, model checking, ...
- Find a **smallest** explanation
  - Debugging, configuration, ...
- Find a **least resource-consuming** schedule
  - Scheduling, logistics, ...
- Find a **most probable** explanation (MAP)
  - Probabilistic inference, ...

High demand for automated approaches to finding good solutions to computationally hard optimization problems

$\rightsquigarrow$ Maximum satisfiability

# Importance of Exact Optimization

## Giving Up?

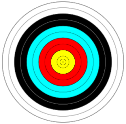
"The problem is NP-hard, so let's develop heuristics / approximation algorithms."

**$$$**

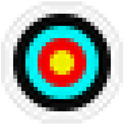
## No!

Benefits of provably optimal solutions:

- Resource savings
  - Money
  - Human resources
  - Time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations



vs



# Importance of Exact Optimization

## Giving Up?

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

## No!

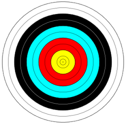
Benefits of provably optimal solutions:

- Resource savings
  - Money
  - Human resources
  - Time
- Accuracy
- Better approximations
  - by optimally solving simplified problem representations

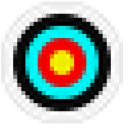
## Key Challenge: Scalability

*Exactly* solving instances of *NP-hard* optimization problems

**$$$**

vs



# Constrained Optimization

Declarative approaches to exact optimization

## Model + Solve

1. **Modeling**:
   represent the problem declarative in a constraint language

   *so that optimal solutions to the constraint model corresponds to optimal solutions of your problem*

2. **Solving**:
   use an generic, exact solver for the constraint language

   *to obtain, for any instance of your problem, an optimal solution to the instance*

# Constrained Optimization

Declarative approaches to exact optimization

## Model + Solve

1. **Modeling**:
   represent the problem declarative in a constraint language

   *so that optimal solutions to the constraint model corresponds to optimal solutions of your problem*
2. **Solving**:
   use an generic, exact solver for the constraint language

   *to obtain, for any instance of your problem, an optimal solution to the instance*

### Important aspects

- Which constraint language to choose — *application-specific*
- How to model the problem compactly & "well" (for the solver)
- Which constraint optimization solver to choose

# Constrained Optimization Paradigms

## Mixed Integer-Linear Programming — MIP, ILP

- Constraint language:
  Conjunctions of linear inequalities $\sum_{i=1}^{k} c_i x_i$
- Algorithms: e.g. Branch-and-cut w/Simplex

## Finite-domain Constraint Optimization — COP

- Constraint language:
  Conjunctions of high-level (global) finite-domain constraints
- Algorithms:
  Depth-first backtracking search, specialized filtering algorithms

## Maximum satisfiability — MaxSAT

- Constraint language:
  weighted Boolean combinations of binary variables
- Algorithms: building on state-of-the-art CDCL SAT solvers
  - Learning from conflicts, conflict-driven search
  - Incremental API, providing explanations for unsatisfiability

# MaxSAT Applications

Drastically increasing number of successful applications

- Planning, Scheduling, and Configuration
- Data Analysis and Machine Learning
- Knowledge Representation and Reasoning
- Combinatorial Optimization
- Verification and Security
- Bioinformatics
- ...
    - *Tens of new problem domains in MaxSAT Evaluations*

# MaxSAT Applications

## Planning, Scheduling, and Configuration

Cost-optimal planning

[Zhang and Bacchus, 2012; Muise, Beck, and McIlraith, 2016]

robot motion planning [Dimitrova, Ghasemi, and Topcu, 2018]

course timetabling

[Demirovic and Musliu, 2017; Manyà, Negrete, Roig, and Soler, 2017; Achá and Nieuwenhuis, 2014]

staff scheduling

[Demirović, Musliu, and Winter, 2017; Bofill, Garcia, Suy, and Villaret, 2015; Cohen, Huang, and Beck, 2017]

vehicle configuration [Marcel Kevin and Tilak Raj, 2016]

package upgradeability

[Argelich, Lynce, and Marques-Silva, 2009; Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010; Ignatiev, Janota, and Marques-Silva, 2014]

...

# MaxSAT Applications

## Data Analysis and Machine Learning

MPE [Park, 2002]

structure learning

[Berg, Järvisalo, and Malone, 2014; Saikko, Malone, and Järvisalo, 2015]

causal discovery [Hyttinen, Saikko, and Järvisalo, 2017b]

causal structure estimation from time series data

[Hyttinen, Plis, Järvisalo, Eberhardt, and Danks, 2017a]

learning explainable decision sets

[Ignatiev, Pereira, Narodytska, and Marques-Silva, 2018b]

interpretable classification rules [Maliotov and Meel, 2018]

constrained correlation clustering [Berg and Järvisalo, 2013, 2017]

neighborhood-preserving visualization

[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]

...

# MaxSAT Applications

## Further AI Applications

dynamics of argumentation

[Wallner, Niskanen, and Järvisalo, 2017; Niskanen, Wallner, and Järvisalo, 2016b,a]

model-based diagnosis [Marques-Silva, Janota, Ignatiev, and Morgado, 2015]

inconsistency analysis

[Lynce and Marques-Silva, 2011; Morgado, Liffiton, and Marques-Silva, 2013b]

...

## Combinatorial Optimization

Max-Clique

[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]

Steiner tree [de Oliveira and Silva, 2015]

tree-width [Berg and Järvisalo, 2014]

maximum quartet consistency [Morgado and Marques-Silva, 2010]

...

# MaxSAT Applications

## Verification and Security

Debugging [Safarpour, Mangassarian, Veneris, Liffiton, and Sakallah, 2007; Chen, Safarpour, Veneris, and Marques-Silva, 2009; Chen, Safarpour, Marques-Silva, and Veneris, 2010; Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b; Xu, Rutenbar, and Sakallah, 2003]

user authorization [Wickramaarachchi, Qardaji, and Li, 2009]

reconstructing AES key schedule images [Liao, Zhang, and Koshimura, 2016]

detecting hardware Trojans [Shabani and Alizadeh, 2018]

malware detection [Feng, Bastani, Martins, Dillig, and Anand, 2017]

QoS [Wakrime, Jabbour, and Hameurlain, 2018; Belabed, Aïmeur, Chikh, and Fethallah, 2017]

program analysis [Mangal, Zhang, Nori, and Naik, 2015; Si, Zhang, Grigore, and Naik, 2017; Zhang, Mangal, Nori, and Naik, 2016]

fault localization [Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]

...

# MaxSAT Applications

## Bioinformatics

Haplotype inference

[Graça, Marques-Silva, and Lynce, 2011a; Graça, Marques-Silva, Lynce, and Oliveira, 2011b]

generalized Ising models

[Huang, Kitchaev, Dacek, Rong, Urban, Cao, Luo, and Ceder, 2016]

bionetworks [Guerra and Lynce, 2012]

cancer therapy design [Lin and Khatri, 2012]

maximum compatibility in phylogenetics [Korhonen and Järvisalo, 2020]

...

# MaxSAT Applications

## Bioinformatics

Haplotype inference

[Graça, Marques-Silva, and Lynce, 2011a; Graça, Marques-Silva, Lynce, and Oliveira, 2011b]

generalized Ising models

[Huang, Kitchaev, Dacek, Rong, Urban, Cao, Luo, and Ceder, 2016]

bionetworks [Guerra and Lynce, 2012]

cancer therapy design [Lin and Khatri, 2012]

maximum compatibility in phylogenetics [Korhonen and Järvisalo, 2020]

...

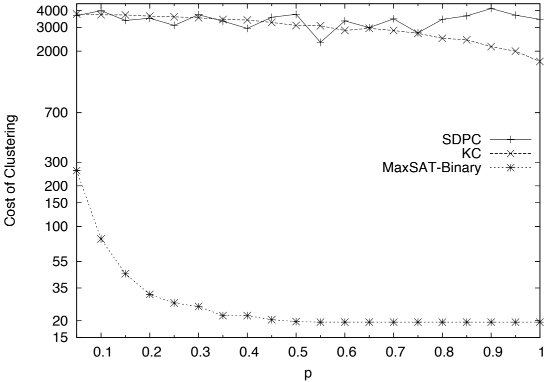
Central to the increasing success:
Advances in MaxSAT solver technology

# Benefits of MaxSAT

Provably optimal solutions

## Example: Correlation clustering by MaxSAT

[Berg and Järvisalo, 2017]




- Improved solution costs over approximative algorithms
- Good performance even on sparse data (missing values)

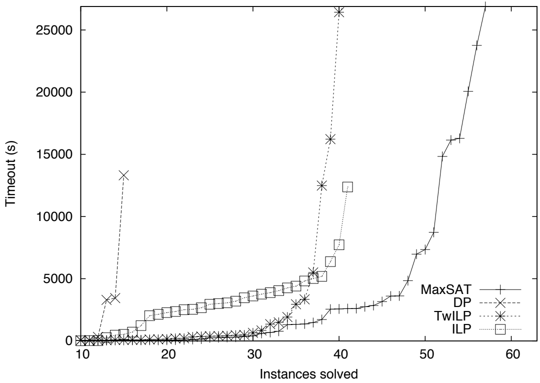
# Benefits of MaxSAT

Surpassing the efficiency of specialized algorithms

Example:
Learning optimal bounded-treewidth Bayesian networks

[Berg, Järvisalo, and Malone, 2014]

MaxSAT vs Dynamic Programming and MIP



# Basic Concepts

# MaxSAT: Basic Definitions

- Simple constraint language:
  conjunctive normal form (CNF) propositional formulas
  - More high-level constraints encoded as sets of clauses
    *Less restrictive than appears—more on this later!*
- Literal: a boolean variable $x$ or $\neg x$.
- Clause $C$: a disjunction ($\vee$) of literals. e.g $(x \vee y \vee \neg z)$
- Truth assignment $\tau$: a function from Boolean variables to $\{0, 1\}$.
- Satisfaction:
  $\tau(C) = 1$ if
  $\tau(x) = 1$ for some literal $x \in C$, or
  $\tau(x) = 0$ for some literal $\neg x \in C$.

  *At least one literal of $C$ is made true by $\tau$.*

# MaxSAT: Basic Definitions

### MaxSAT

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

TASK: find $\tau$ s.t. $\sum_{C \in F} \tau(C)$ is maximized.

Find truth assignment that satisfies a maximum number of clauses

This is the standard definition, much studied in Theoretical Computer Science.

- Often inconvenient for modeling practical problems.

# Central Generalizations of MaxSAT

## Weighted MaxSAT

- Each clause $C$ has an associated weight $w_C$
- Optimal solutions maximize the sum of *weights* of satisfied clauses: $\tau$ s.t. $\sum_{C \in F} w_c \tau(C)$ is maximized.

## Partial MaxSAT

- Some clauses are deemed *hard*—infinite weights
  - Any solution has to satisfy the hard clauses
    $\rightsquigarrow$ Existence of solutions not guaranteed
- Clauses with finite weight are *soft*

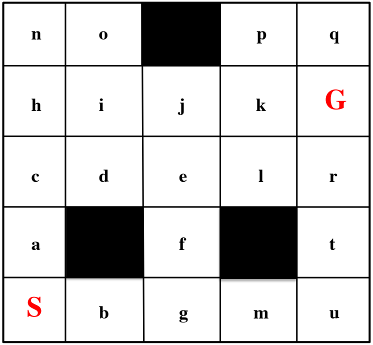
## Weighted Partial MaxSAT

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

# MaxSAT: Example

## Shortest Path

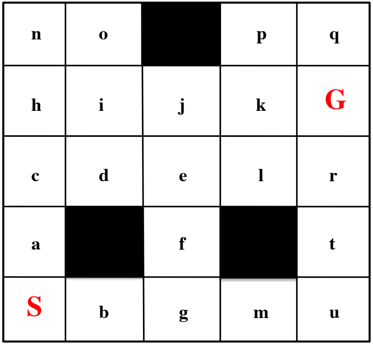
Find shortest path in a grid with horizontal/vertical moves.
Travel from S to G.
Cannot enter blocked squares.



# MaxSAT: Example

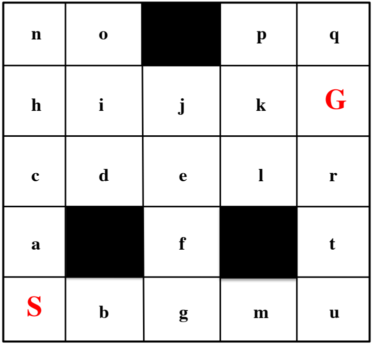
- Note: Best solved with state-space search
  - Used here to illustrate MaxSAT encodings

# MaxSAT: Example




- Boolean variables: one for each unblocked grid square $\{S, G, a, b, \ldots, u\}$: $\text{true}$ *iff path visits this square.*

# MaxSAT: Example




- Boolean variables: one for each unblocked grid square $\{S, G, a, b, \ldots, u\}$: true *iff path visits this square.*
- Constraints:
  - The S and G squares must be visited: In CNF: unit hard clauses $(S)$ and $(G)$.
  - A soft clause of weight 1 for all other squares: In CNF: $(\neg a)$, $(\neg b)$, …, $(\neg u)$ "*would prefer not to visit*"

# MaxSAT: Example

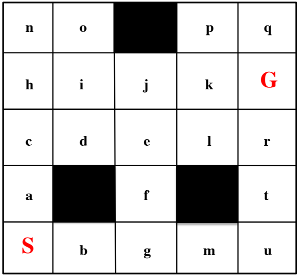
- The previous clauses minimize the number of visited squares.
- ...however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

# MaxSAT: Example

- The previous clauses minimize the number of visited squares.
- ...however, their MaxSAT solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

## A way to enforce a path between S and G:

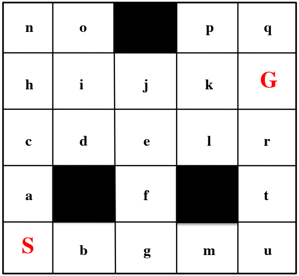
- both S and G must have *exactly one* visited neighbour
  - Any path starts from S
  - Any path ends at G
- other visited squares must have *exactly two* visited neighbours
  - One predecessor and one successor on the path



# MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

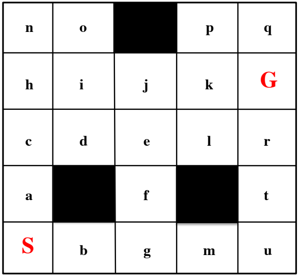


# MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

- For S: $a + b = 1$
  - In CNF: $(a \vee b), (\neg a \vee \neg b)$



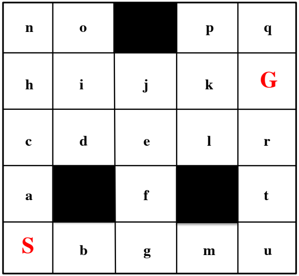
# MaxSAT: Example

Constraint 1:

*S and G must have exactly one visited neighbour.*

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

  *disallow pairwise*

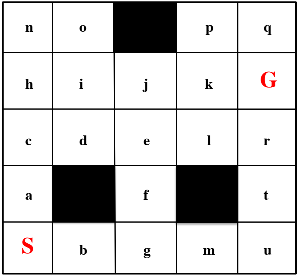


# MAXSAT: Example

Constraint 2:

*Other visited squares must have exactly two visited neighbours*

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



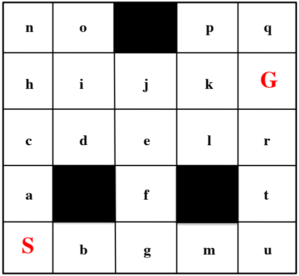
# MaxSAT: Example

Constraint 2:
*Other visited squares must have exactly two visited neighbours*

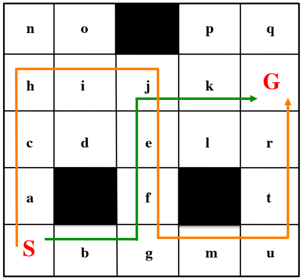
- For example, for square $e$: $e \rightarrow (d + j + l + f = 2)$
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

## Encoding Cardinality Constraints in CNF

- An important class of constraints, occur frequently in real-world problems
- A lot of work on CNF encodings of cardinality constraints



# MAXSAT: Example




## Properties of the encoding

- Every solution to the hard clauses is a path from S to G that does not pass a blocked square.
- Such a path will falsify one negative soft clause for every square it passes through.
  - orange path: assign 14 variables in $\{S, a, c, h, \ldots, t, r, G\}$ to true
- MAXSAT solutions:
  paths that pas through a minimum number of squares (i.e., is shortest).
  - green path: assign 8 variables in $\{S, b, g, f, \ldots, k, G\}$ to true

# MaxSAT: Complexity

## Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.
**Question:**
Is there an assignment that satisfies at least $k$ clauses in $F$?

# MaxSAT: Complexity

## Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.
**Question:**
Is there an assignment that satisfies at least $k$ clauses in $F$?

## MaxSAT is $\mathrm{FP}^{\mathrm{NP}}$–complete

- The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle
  - Polynomial number of oracle calls
  - Other $\mathrm{FP}^{\mathrm{NP}}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

# MaxSAT: Complexity

## Deciding whether $k$ clauses can be satisfied: NP-complete

**Input:** A CNF formula $F$, a positive integer $k$.
**Question:**
Is there an assignment that satisfies at least $k$ clauses in $F$?

## MaxSAT is $FP^{NP}$–complete

- The class of binary relations $f(x, y)$ where given $x$ we can compute $y$ in polynomial time with access to an NP oracle
  - Polynomial number of oracle calls
  - Other $FP^{NP}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

## MaxSAT is hard to approximate APX–complete

APX: class of NP optimization problems that

- admit a constant-factor approximation algorithm, *but*
- have no poly-time approximation scheme (unless NP=P).

# Push-Button Solvers

- Black-box, *no command line parameters necessary*
- Input: CNF formula, in the *standard* DIMACS WCNF file format
- Output: provably optimal solution, or UNSATISFIABLE
  - Complete solvers

mancoosi-test-i2000d0u98-26.wcnf

```
p wcnf 18169 112632 31540812410
31540812410 -1 2 3 0
31540812410 -4 2 3 0
31540812410 -5 6 0
...
18170 1133 0
18170 457 0
```

... truncated 2.4 MB

Internally rely especially on CDCL SAT solvers

*for proving unsatisfiability of subsets of clauses*

# Push-Button Solver Technology

Example: `$ openwbo mancoosi-test-i2000d0u98-26.wcnf`

```
c Open-WBO: a Modular MaxSAT Solver
c Version: 1.3.1 – 18 February 2015
...
c | Problem Type: Weighted
c | Number of variables: 18169
c | Number of hard clauses: 94365
c | Number of soft clauses: 18267
c | Parse time: 0.02 s
...
o 10548793370
c LB : 15026590
c Relaxed soft clauses 2 / 18267
c LB : 30053180
c Relaxed soft clauses 3 / 18267
c LB : 45079770
c Relaxed soft clauses 5 / 18267
c LB : 60106360
...
c Relaxed soft clauses 726 / 18267
c LB : 287486453
c Relaxed soft clauses 728 / 18267
o 287486453
c Total time: 1.30 s
c Nb SAT calls: 4
c Nb UNSAT calls: 841
s OPTIMUM FOUND
v 1 -2 3 4 5 6 7 8 -9 10 11 12 13 14 15 16 ...
... -18167 -18168 -18169 -18170
```

# Standard Solver Input Format: DIMACS WCNF

- Variables indexed from 1 to $n$
- Negation: -
  - -3 stand for $\neg x_3$
- 0: special end-of-line character
- One special header "p"-line:
  `p wcnf <#vars> <#clauses> <top>`
  - `#vars`: number of variables $n$
  - `#clauses`: number of clauses
  - `top`: "weight" of *hard* clauses.
    - *Any number larger than the sum of soft clause weights can be used.*
- Clauses represented as lists of integers
  - Weight is the first number
  - $(-x_3 \vee x_1 \vee \neg x_{45})$, weight 2:
    `2 -3 1 -45 0`
- Clause is hard if weight == `top`

Example:

mancoosi-test-i2000d0u98-26.wcnf

```
p wcnf 18169 112632 31540812410
31540812410 -1 2 3 0
31540812410 -4 2 3 0
31540812410 -5 6 0
...
18170 1133 0
18170 457 0
```

... truncated 2.4 MB

# MaxSAT Evaluations

`https://maxsat-evaluations.github.io`

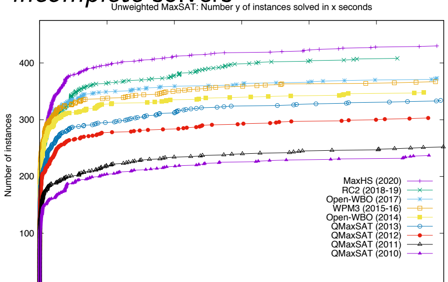
## Objectives

- Assessing the state of the art in the field of MaxSAT solvers
- Collecting publicly available MaxSAT benchmark sets
- *Tens of solvers from various research groups internationally* participate each year
- Standard input format
- Tracks for both *complete* and *incomplete* solvers

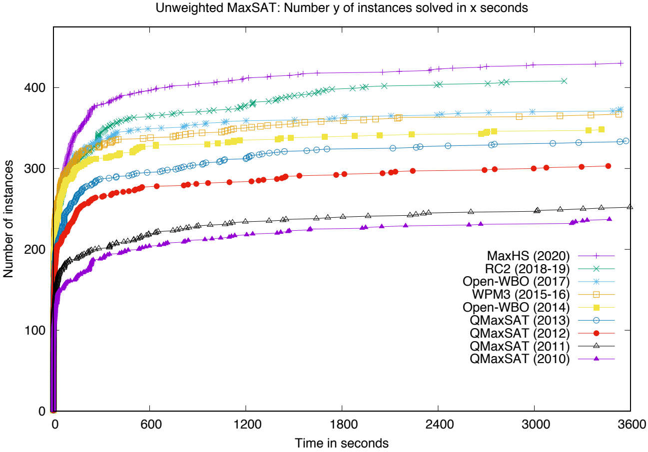
**15th MaxSAT Evaluation**

`https://maxsat-evaluations.github.io/2020`

Affiliated with SAT 2020 conference



# Progress in MaxSAT Solver Performance




## Comparing some of the best solvers from 2010–2020:

In 2020: 81% more instances solved than in 2010!

- On same computer, same set of benchmarks:
  576 unweighted MaxSAT Evaluation 2020 instances

# MaxSAT Solving: Practical Algorithms for MaxSAT

# Types of MaxSAT Solvers

### MaxSAT Solver

Practical implementation of an algorithm for finding (optimal) solutions to MaxSAT instances

### Complete vs Incomplete MaxSAT Solvers

- Complete:
  *Guaranteed to output a provably optimal solution to any instance*
  *(given enough resources (time & space))*

- “Incomplete”:
  *Tailored to provide “good” solutions quickly*
  *(potentially) no guarantees on optimality of solutions*

# Availability: Some Recent MaxSAT Solvers

Examples of recent solvers

## Complete

- RC2 `https://pysathq.github.io/docs/html/api/examples/rc2.html`
- Maxino `https://alviano.net/software/maxino`
- UWrMaxSAT `https://github.com/marekpiotrow/UWrMaxSat`
- OpenWBO `http://sat.inesc-id.pt/open-wbo`
- MaxHS `http://maxhs.org`
- QMaxSAT `https://sites.google.com/site/qmaxsat`

## Incomplete

- Loandra `https://github.com/jezberg/loandra`
- Open-WBO-Inc `https://github.com/sbjoshi/Open-WBO-Inc`
- Open-WBO-TT `http://www.cs.tau.ac.il/research/alexander.nadel`
- SATLike `http://lcs.ios.ac.cn/~caisw/MaxSAT.html`

# Availability

## Open Source

Starting from 2017, solvers need to be open-source in order to participate in MaxSAT Evaluations

- Incentive for openness
- Allow other to build on and test new ideas on establish solver source bases

`https://maxsat-evaluations.github.io/`

# Complete MaxSAT Solving

# Types of Complete Solvers

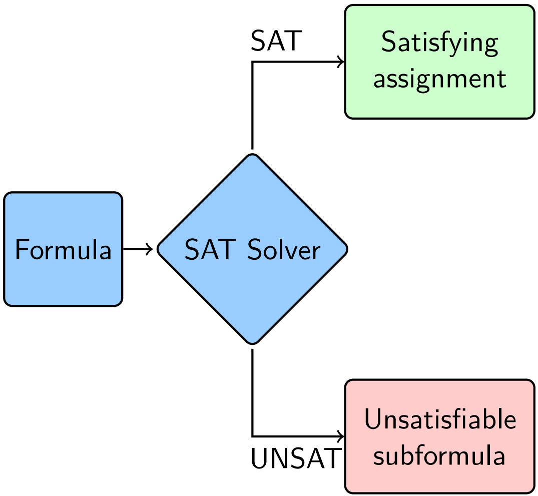
- Branch and Bound
  - Can be effective of small-but hard & randomly generated instances
- SAT-based MaxSAT algorithms
  - Model-improving
  - Core-guided
  - Implicit hitting set

Focus here: SAT-based MaxSAT solving

- Make use of iterative SAT solver calls
- Key to solving very large real-world problem instances as MaxSAT

# SAT-based MaxSAT Algorithms

# SAT Solvers


Formula
SAT Solver
SAT
Satisfying assignment
UNSAT
Unsatisfiable subformula

# Satisfying assignment

Formula:

$$x_1 \qquad x_2 \vee \neg x_1 \qquad \neg x_3 \vee x_1 \qquad \neg x_3 \vee \neg x_1 \qquad x_2 \vee \neg x_3$$

- Satisfying assignment:
  - Assignment to the variables that evaluates the formula to true

# Satisfying assignment

Formula:

$$x_1 \qquad x_2 \vee \neg x_1 \qquad \neg x_3 \vee x_1 \qquad \neg x_3 \vee \neg x_1 \qquad x_2 \vee \neg x_3$$

- Satisfying assignment:
  - Assignment to the variables that evaluates the formula to true
  - $\tau = \{x_1 = 1, x_2 = 1, x_3 = 0\}$

# Unsatisfiable subformula — UNSAT Cores

Formula:

$$x_1 \qquad x_3 \qquad x_2 \vee \neg x_1 \qquad \neg x_3 \vee x_1 \qquad \neg x_2 \vee \neg x_1 \qquad x_2 \vee \neg x_3$$

- Formula is unsatisfiable

# Unsatisfiable subformula — UNSAT Cores

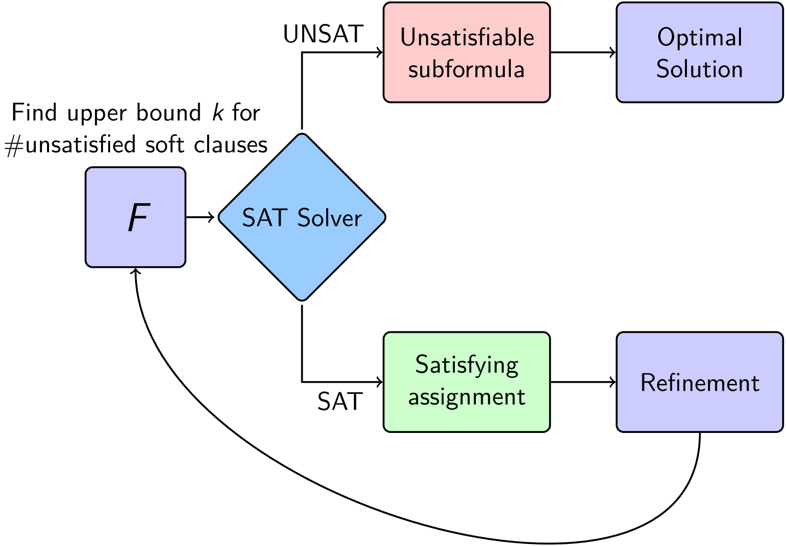
Formula:

$$x_1 \quad x_3 \quad x_2 \vee \neg x_1 \quad \neg x_3 \vee x_1 \quad \neg x_2 \vee \neg x_1 \quad x_2 \vee \neg x_3$$

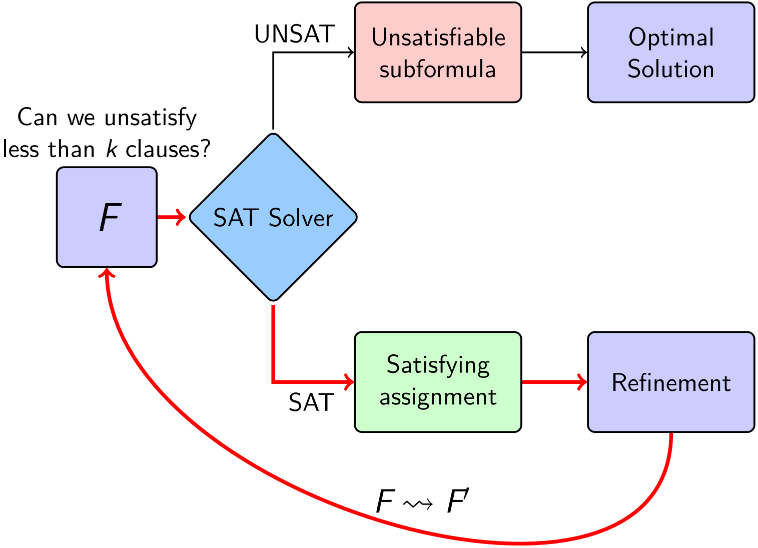
- Formula is unsatisfiable
- Unsatisfiable subformula (core):
  - $F' \subseteq F$, such that $F'$ is unsatisfiable

# Model-Improving MaxSAT

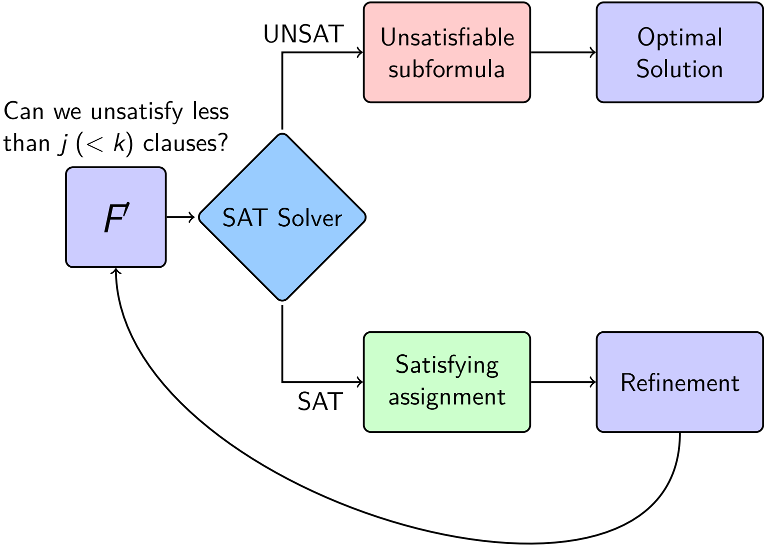
# Upper Bound Search for MaxSAT



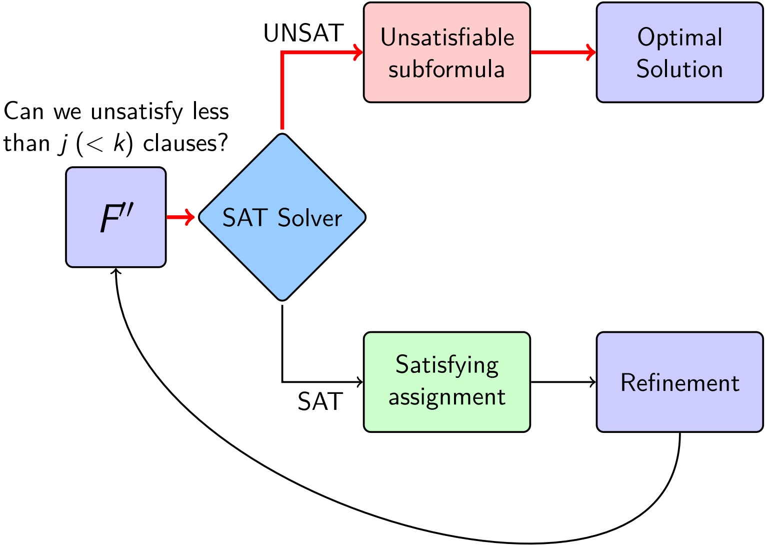
# Upper Bound Search for MaxSAT



# Upper Bound Search for MaxSAT



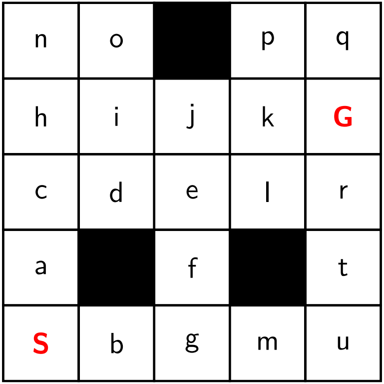
# Upper Bound Search for MaxSAT



# Model-Improving Algorithm

## Shortest Path

### Intuition



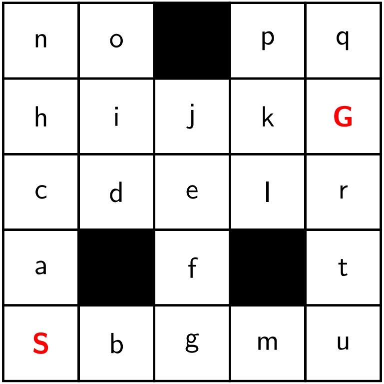

$\mathrm{UB} = \infty$

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$




$\mathrm{UB} = \infty$

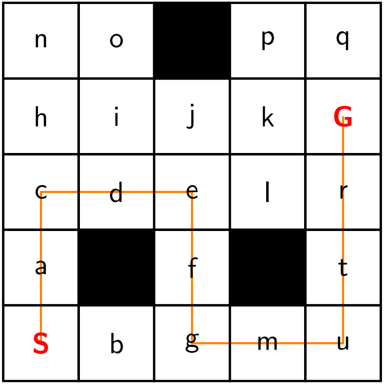
SAT-SOLVE(**H**)

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$
2. Update UB




$\mathrm{UB} = \mathbf{10}$

$\textsc{SAT-SOLVE}(H)$

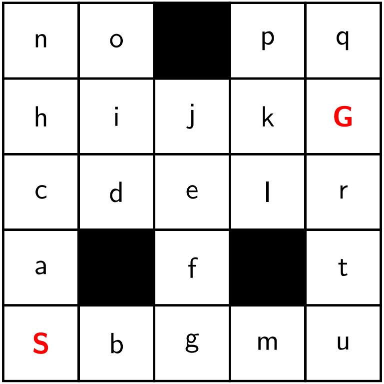
$\tau^1 = \{S, a, c, d, e, f, g, m, u, t, r, G, \neg b, \neg l, \ldots, \neg q\}$
$cost(\tau^1) = 10$

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$
2. Update UB
3. Improve $\tau^*$ until $\tau^*$ is proven to be optimal




$\mathrm{UB} = 10$

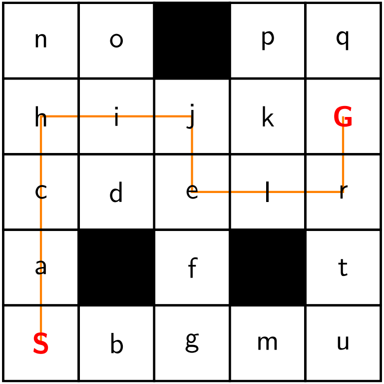
$\text{SAT-SOLVE}\,(\mathbf{H} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{UB}))$

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$
2. Update UB
3. Improve $\tau^*$ until $\tau^*$ is proven to be optimal




$\mathrm{UB} = \mathbf{8}$

$\mathrm{SAT\text{-}SOLVE}\,(H \wedge \textsc{CostLessThan}(S, UB))$

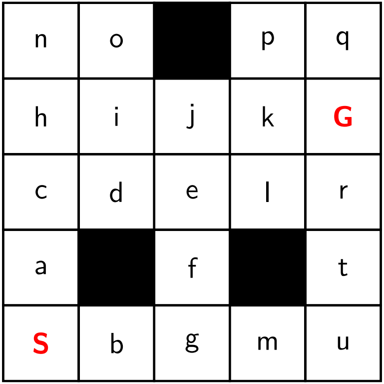
$\tau^2 = \{S, a, c, h, i, j, e, l, r, G, \neg b, \neg g, \ldots, \neg q\}$
$cost(\tau^2) = 8$

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$
2. Update UB
3. Improve $\tau^*$ until $\tau^*$ is proven to be optimal




$\mathrm{UB} = 8$

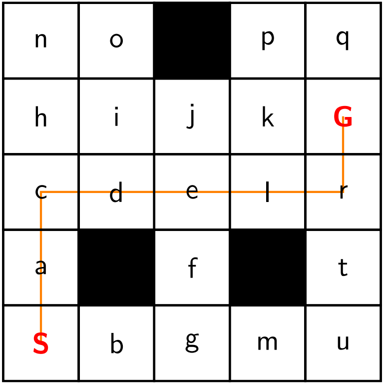
$\text{SAT-SOLVE}\,(\mathbf{H} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{UB}))$

# Model-Improving Algorithm

## Shortest Path

### Intuition

1. Obtain a solution $\tau^*$
2. Update UB
3. Improve $\tau^*$ until $\tau^*$ is proven to be optimal




$\mathrm{UB} = \mathbf{6}$

$\mathrm{SAT\text{-}SOLVE}\,(H \wedge \textsc{CostLessThan}(S, UB))$

$\tau^3 = \{S, a, c, d, e, l, r, G, \neg b, \neg g, \ldots, \neg q\}$

$cost(\tau^3) = 6$

# Model-Improving Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$ (Soft): | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

# Model-Improving Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

$H:$ $\neg x_2 \vee \neg x_1$ $\quad x_2 \vee \neg x_3$

$S:$ $x_1 \vee r_1$ $\quad x_3 \vee r_2$ $\quad x_2 \vee \neg x_1 \vee r_3$ $\quad \neg x_3 \vee x_1 \vee r_4$

- Relax all soft clauses
- Relaxation variables:
  - $R = \{r_1, r_2, r_3, r_4\}$
  - If a soft clause $\omega_i$ is **unsatisfied**, then $r_i = 1$
  - If a soft clause $\omega_i$ is **satisfied**, then $r_i = 0$

# Model-Improving Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ : | | $\neg x_2 \lor \neg x_1$ | $x_2 \lor \neg x_3$ | |
| $S$ : | $x_1 \lor r_1$ | $x_3 \lor r_2$ | $x_2 \lor \neg x_1 \lor r_3$ | $\neg x_3 \lor x_1 \lor r_4$ |

$R = \{r_1, r_2, r_3, r_4\}$

- Formula is satisfiable
  - $\tau = \{x_1 = 1, x_2 = 0, x_3 = 0, r_1 = 0, r_2 = 1, r_3 = 1, r_4 = 0\}$
- **Goal:** Minimize number of relaxation variables assigned to 1

# Can we unsatisfy less than 2 soft clauses?

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ : | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$ : | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1 \vee r_3$ | $\neg x_3 \vee x_1 \vee r_4$ |

$cost(\tau) = 2 \qquad R = \{r_1, r_2, r_3, r_4\}$

- $r_2$ and $r_3$ were assigned truth value 1:
  - Current solution unsatisfies 2 soft clauses
- Can less than 2 soft clauses be unsatisfied?

# Can we unsatisfy less than 2 soft clauses?

## Solving at the formula level

Partial MaxSAT Formula:

$H: \quad \neg x_2 \vee \neg x_1 \quad x_2 \vee \neg x_3 \quad \text{CNF}(\sum_{r_i \in R} r_i \leq 1)$

$S: \quad x_1 \vee r_1 \quad x_3 \vee r_2 \quad x_2 \vee \neg x_1 \vee r_3 \quad \neg x_3 \vee x_1 \vee r_4$

$cost(\tau) = 2 \quad R = \{r_1, r_2, r_3, r_4\}$

- Add cardinality constraint that excludes solutions that unsatisfies 2 or more soft clauses:
  - $\text{CNF}(r_1 + r_2 + r_3 + r_4 \leq 1)$

# Can we unsatisfy less than 2 soft clauses? No!

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ : | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(\sum_{r_i \in R} r_i \leq 1)$ | |
| $S$ : | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1 \vee r_3$ | $\neg x_3 \vee x_1 \vee r_4$ |

$cost(\tau) = 2 \quad R = \{r_1, r_2, r_3, r_4\}$

- Formula is unsatisfiable:
  - There are no solutions that unsatisfy 1 or less soft clauses

# Can we unsatisfy less than 2 soft clauses? No!

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

$cost(\tau) = 2 \qquad R = \{r_1, r_2, r_3, r_4\}$

- **Optimal solution**: given by the last model and corresponds to unsatisfying 2 soft clauses:
  - $\tau = \{x_1 = 1, x_2 = 0, x_3 = 0\}$

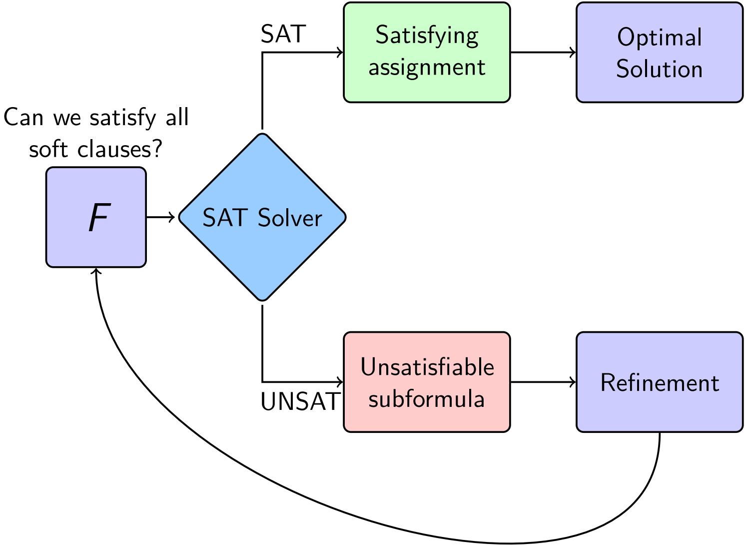
# Model-Improving Algorithm

## Summary

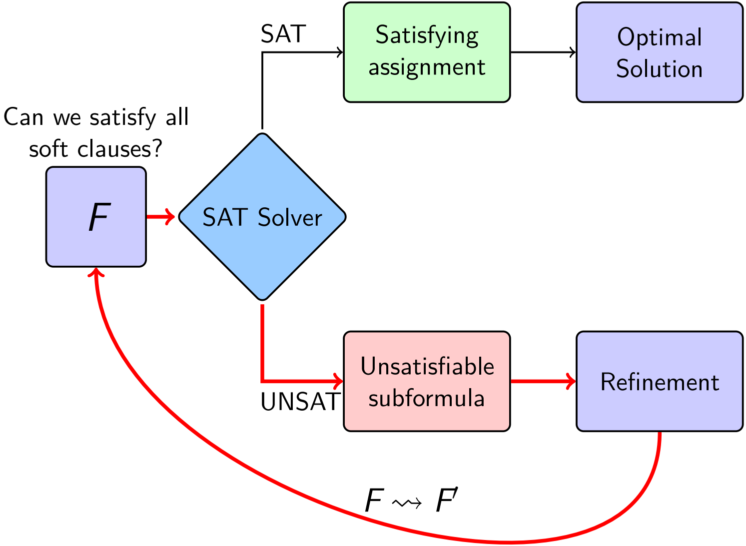
- Model-improving can be very efficient when:
  - The number of soft clauses is small
  - The optimal solution corresponds to unsatisfying the majority of soft clauses
- Example of state-of-the-art solvers that use this algorithm:
  - QMaxSAT [Koshimura, Zhang, Fujita, and Hasegawa, 2012]
  - Pacose [Paxian, Reimer, and Becker, 2018]
- Challenges:
  - Constraint that restricts the UB grows with the number of soft clauses (weights of the soft clauses)
- Alternatives:
  - What other kind of search can we perform?
  - What if we start searching from the lower bound?

# Core-Guided MaxSAT Solving

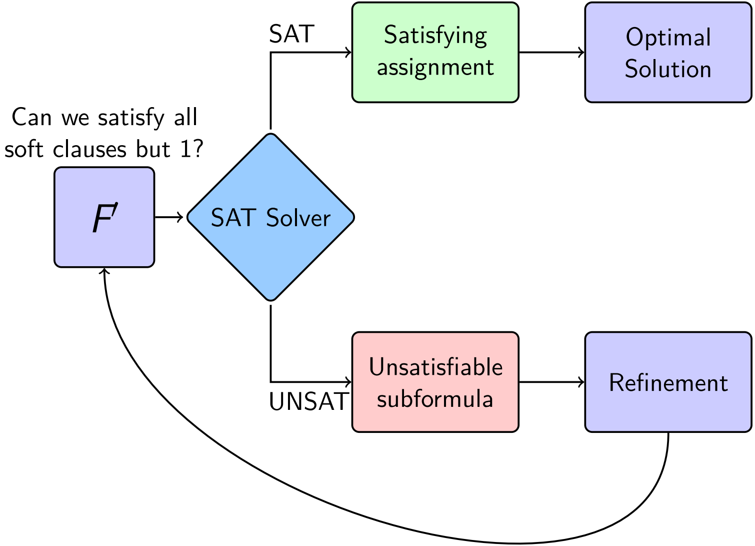
# Lower Bound Search for MaxSAT



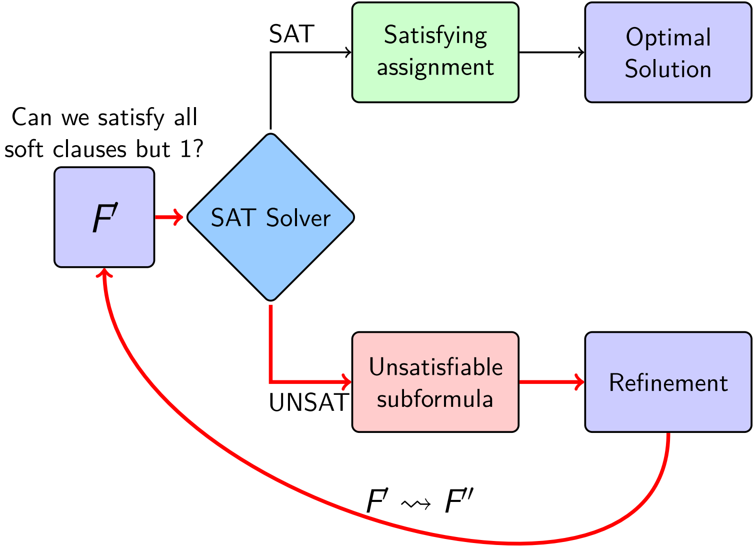
# Lower Bound Search for MaxSAT



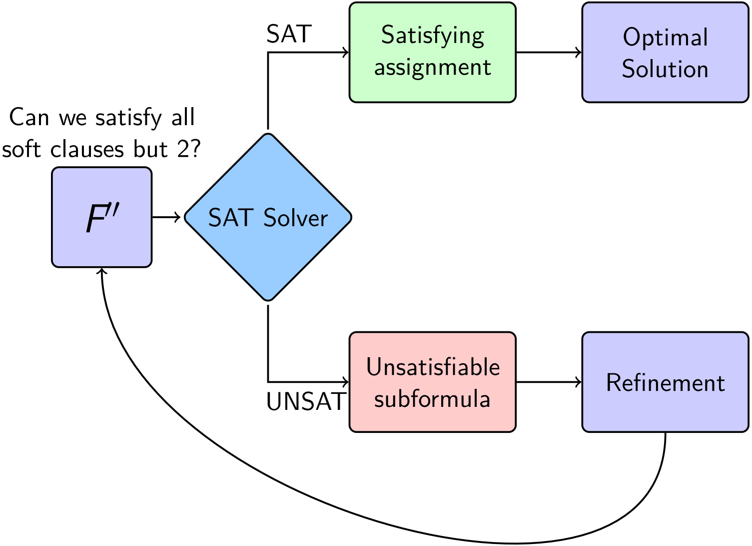
# Lower Bound Search for MaxSAT



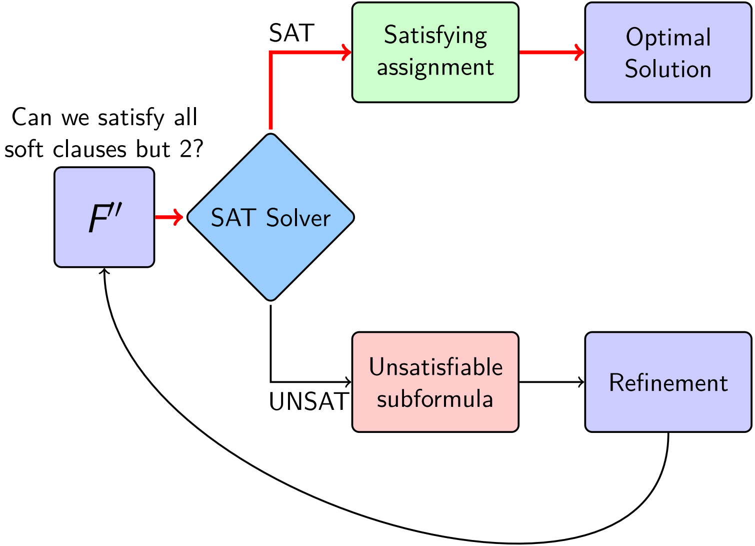
# Lower Bound Search for MaxSAT



# Lower Bound Search for MaxSAT



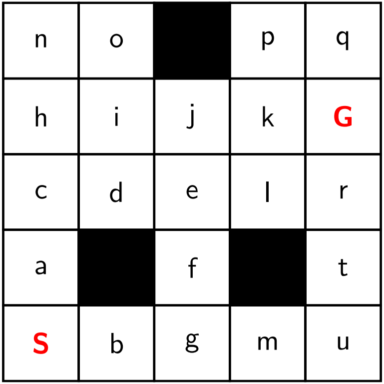
# Lower Bound Search for MaxSAT



# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition




$\mathrm{LB} = 0$

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(S, LB)$ is satisfiable

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

$\mathrm{LB} = 0$

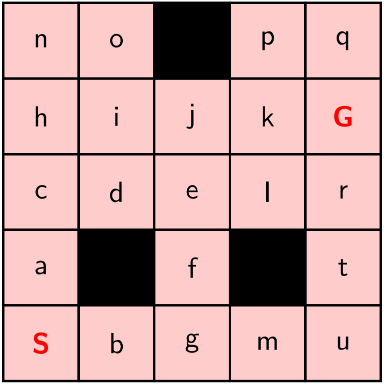
$\mathrm{SAT\text{-}SOLVE}(\mathbf{H} \wedge \mathbf{S} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{LB}))$

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(S, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB




$\mathrm{LB} = 0$

$\text{SAT-SOLVE}(H \wedge S \wedge \textsc{CostLessThan}(S, LB))$

Formula is unsatisfiable

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \land S \land \textsc{CostLessThan}(S, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB

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

$\mathrm{LB} = 1$

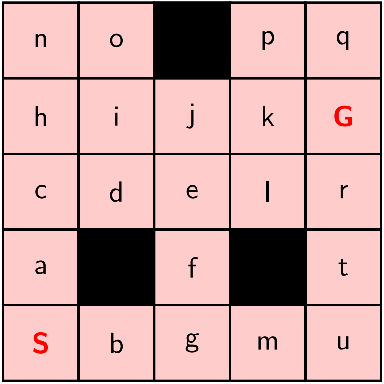
$\text{SAT-SOLVE}\,(\mathbf{H} \land \mathbf{S} \land \textsc{CostLessThan}(\mathbf{S}, \mathbf{LB}))$

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(S, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB




$\mathrm{LB} = 1$

$\text{SAT-SOLVE}\,(H \wedge S \wedge \textsc{CostLessThan}(S, LB))$

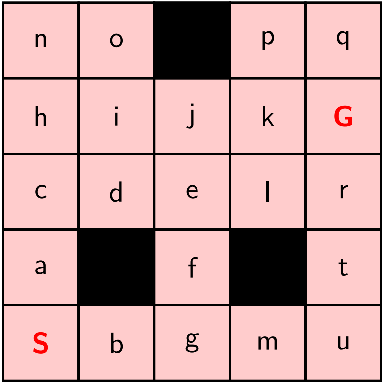
Formula is unsatisfiable

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(S, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB




$\mathrm{LB} = \{\mathbf{2}, \ldots \mathbf{5}\}$

$\mathrm{SAT\text{-}SOLVE}\,(\mathbf{H} \wedge \mathbf{S} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{LB}))$

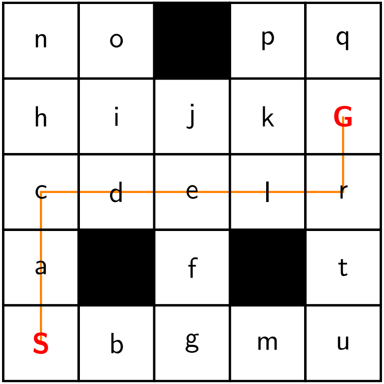
Formula is unsatisfiable

# Unsatisfiability-based Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(S, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB
3. Otherwise, an optimal model $\tau$ has been found




$\mathrm{LB} = \mathbf{6}$

$\mathrm{SAT\text{-}SOLVE}\,(H \wedge S \wedge \textsc{CostLessThan}(S, LB))$

$\tau = \{S, a, c, d, e, l, r, G, \neg b, \neg g, \ldots, \neg q\}$

$cost(\tau) = 6$

# Unsatisfiability-based algorithm

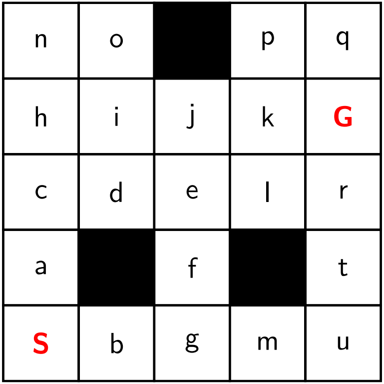
## Summary

- Challenges:
  - Incrementality, i.e. maintaining information across iterations
  - Constraint that restricts the LB grows with the number of soft clauses (weights of the soft clauses)
- No existing solver that uses this algorithm:
  - There exists better unsatisfiability-based algorithms
- Alternatives:
  - Change the refinement procedure to relax soft clauses lazily:
    - Use **unsat cores** to only consider a subset of the soft clauses
    - Constraint that restricts the LB will be much smaller
    - Can scale to problems with millions of soft clauses

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition



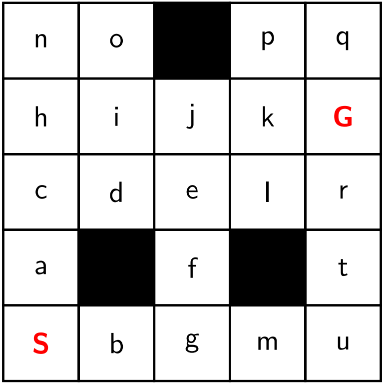

$\mathrm{LB} = 0, R = \{\}$

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable




$\mathrm{LB} = 0, R = \{\}$

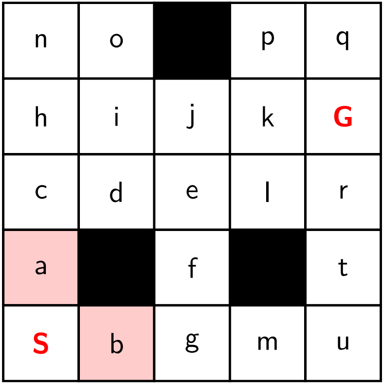
$\text{SAT-SOLVE}(\mathbf{H} \wedge \mathbf{S} \wedge \textsc{CostLessThan}(\mathbf{R}, \mathbf{LB}))$

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB and update $R$




$\mathrm{LB} = 0$

$\text{SAT-SOLVE}(H \wedge S \wedge \textsc{CostLessThan}(R, LB))$

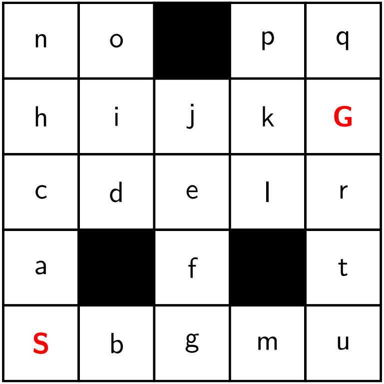
Formula is unsatisfiable
Use unsat core to update $R = \{a, b\}$

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB and update $R$




$\mathrm{LB} = 1, R = \{a, b\}$

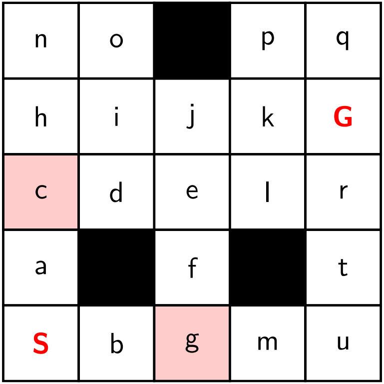
$\textcolor{red}{\text{SAT-SOLVE}\,(\mathbf{H} \wedge \mathbf{S} \wedge \textsc{CostLessThan}(\mathbf{R}, \mathbf{LB}))}$

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB and update $R$




$\mathrm{LB} = 1, R = \{a, b\}$

$\text{SAT-SOLVE}\,(H \wedge S \wedge \textsc{CostLessThan}(R, LB))$

Formula is unsatisfiable
Use unsat core to update $R = \{a, b, c, g\}$

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB and update $R$

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

$\mathrm{LB} = \{\mathbf{2}, \ldots \mathbf{5}\}, R = \{a, b, c, g, \ldots\}$

$\text{SAT-SOLVE}\,(\mathbf{H} \wedge \mathbf{S} \wedge \textsc{CostLessThan}(\mathbf{R}, \mathbf{LB}))$

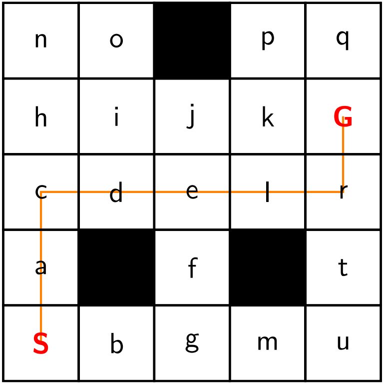
Formula is unsatisfiable

# MSU3 Core-Guided Algorithm

## Shortest Path

### Intuition

1. Check if $H \wedge S \wedge \textsc{CostLessThan}(R, LB)$ is satisfiable
2. If it is unsatisfiable, then increase LB and update $R$
3. Otherwise, an optimal model $\tau$ has been found




$\mathrm{LB} = \mathbf{6}, R = \{a, b, \ldots\}$

$\textsc{SAT-Solve}\,(H \wedge S \wedge \textsc{CostLessThan}(R, LB))$

$\tau = \{S, a, c, d, e, l, r, G, \neg b, \neg g, \ldots, \neg q\}$

$cost(\tau) = 6$

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$ (Soft): | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable
- Identify an unsatisfiable core

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

$H$: $\neg x_2 \vee \neg x_1$ $\quad x_2 \vee \neg x_3$ $\quad \text{CNF}(r_1 + r_2 \leq 1)$

$S$: $x_1 \vee r_1$ $\quad x_3 \vee r_2$ $\quad x_2 \vee \neg x_1$ $\quad \neg x_3 \vee x_1$

- Relax non-relaxed soft clauses in unsatisfiable core:
  - Add cardinality constraint that excludes solutions that unsatisfies 2 or more soft clauses:
    - $\text{CNF}(r_1 + r_2 \leq 1)$
  - Relaxation on demand instead of relaxing all soft clauses eagerly

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(r_1 + r_2 \leq 1)$ | |
| $S$: | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(r_1 + r_2 \leq 1)$ | |
| $S$: | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable
- Identify an unsatisfiable core

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

$H$: $\neg x_2 \vee \neg x_1$ $\quad x_2 \vee \neg x_3$ $\quad \boxed{\mathrm{CNF}(r_1 + \ldots + r_4 \leq 2)}$

$S$: $x_1 \vee r_1$ $\quad x_3 \vee r_2$ $\quad x_2 \vee \neg x_1 \vee \boxed{r_3}$ $\quad \neg x_3 \vee x_1 \vee \boxed{r_4}$

- Relax non-relaxed soft clauses in unsatisfiable core:
  - Add cardinality constraint that excludes solutions that unsatisfies 3 or more soft clauses:
    - $\mathrm{CNF}(r_1 + r_2 + r_3 + r_4 \leq 2)$
  - Relaxation on demand instead of relaxing all soft clauses eagerly

# MSU3 Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ : | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(r_1 + \ldots + r_4 \leq 2)$ | |
| $S$ : | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1 \vee r_3$ | $\neg x_3 \vee x_1 \vee r_4$ |

- Formula is satisfiable:
  - $\tau = \{x_1 = 1, x_2 = 0, x_3 = 0, r_1 = 0, r_2 = 1, r_3 = 1, r_4 = 0\}$
- Optimal solution unsatisfies 2 soft clauses

# MSU3 Core-Guided Algorithm

## Summary

- MSU3 algorithm can be very efficient when:
  - The size of the cores found at each iteration are small
  - The optimal solution corresponds to satisfying the majority of soft clauses
- Example of state-of-the-art solvers that use this algorithm:
  - Open-WBO [Martins, Manquinho, and Lynce, 2014b]
- Challenges:
  - Constraint that restricts the LB grows with the size of cores
  - Does not capture local core information:
    - In 2nd iteration for the shortest path example MSU3 used the cardinality constraint: $(r_a + r_b + r_c + r_g \leq 2)$
    - But at this stage we actually know something stronger: $(r_a + r_b \leq 1)$ and $(r_c + r_g \leq 1)$
- Alternatives:
  - Fu-Malik algorithm encodes each core separately by relaxing each soft clause multiple times

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$ (Soft): | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable
- Identify an unsatisfiable core

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

$H$: $\neg x_2 \vee \neg x_1$ $\quad x_2 \vee \neg x_3$ $\quad \boxed{\mathrm{CNF}(r_1 + r_2 \leq 1)}$

$S$: $x_1 \vee \boxed{r_1}$ $\quad x_3 \vee \boxed{r_2}$ $\quad x_2 \vee \neg x_1$ $\quad \neg x_3 \vee x_1$

- Relax unsatisfiable core:
  - Add relaxation variables
  - Add AtMost1 constraint

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\mathrm{CNF}(r_1 + r_2 \leq 1)$ | |
| $S$: | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(r_1 + r_2 \leq 1)$ | |
| $S$: | $x_1 \vee r_1$ | $x_3 \vee r_2$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is unsatisfiable
- Identify an unsatisfiable core

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

$H$: $\neg x_2 \vee \neg x_1$ $\quad x_2 \vee \neg x_3$ $\quad \mathrm{CNF}(r_1 + r_2 \leq 1)$ $\quad \boxed{\mathrm{CNF}(r_3 + \ldots + r_6 \leq 1)}$

$S$: $x_1 \vee r_1 \vee \boxed{r_3}$ $\quad x_3 \vee r_2 \vee \boxed{r_4}$ $\quad x_2 \vee \neg x_1 \vee \boxed{r_5}$ $\quad \neg x_3 \vee x_1 \vee \boxed{r_6}$

- Relax unsatisfiable core:
  - Add relaxation variables
  - Add AtMost1 constraint
- Soft clauses may be relaxed multiple times

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | $\text{CNF}(r_1 + r_2 \leq 1)$ | $\text{CNF}(r_3 + \ldots + r_6 \leq 1)$ |
| $S$: | $x_1 \vee r_1 \vee r_3$ $\quad$ $x_3 \vee r_2 \vee r_4$ | | $x_2 \vee \neg x_1 \vee r_5$ | $\neg x_3 \vee x_1 \vee r_6$ |

- Formula is satisfiable
- An optimal solution would be:
  - $\tau = \{x_1 = 1, x_2 = 0, x_3 = 0\}$

# Fu-Malik Core-Guided Algorithm

## Solving at the formula level

Partial MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$: | | $\neg x_2 \vee \neg x_1$ | $x_2 \vee \neg x_3$ | |
| $S$: | $x_1$ | $x_3$ | $x_2 \vee \neg x_1$ | $\neg x_3 \vee x_1$ |

- Formula is satisfiable
- An optimal solution would be:
  - $\tau = \{x_1 = 1, x_2 = 0, x_3 = 0\}$
- This assignment unsatisfies 2 soft clauses

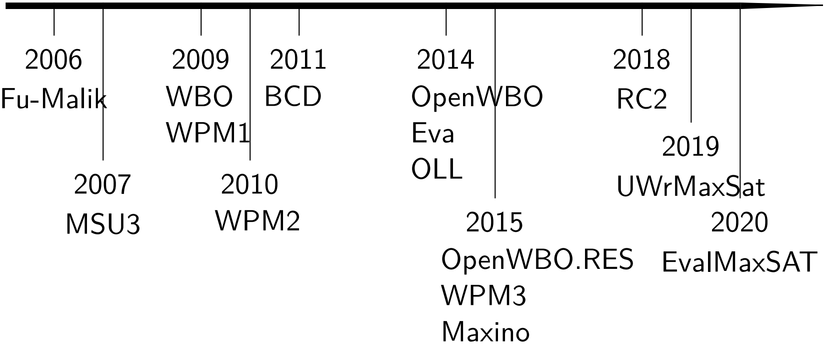
# Fu-Malik Core-Guided Algorithm

## Summary

- Encoding cardinality constraints into CNF is efficient since it only uses AtMost 1 constraints
- Previous MaxSAT solvers that used this algorithm:
  - WBO [Manquinho, Marques-Silva, and Planes, 2009]
  - WPM1 [Ansótegui, Bonet, and Levy, 2009]
- Challenges:
  - Number of relaxation variables per soft clause can grow significantly
  - Multiple cardinality constraints

# Core-Guided Algorithms

## Timeline

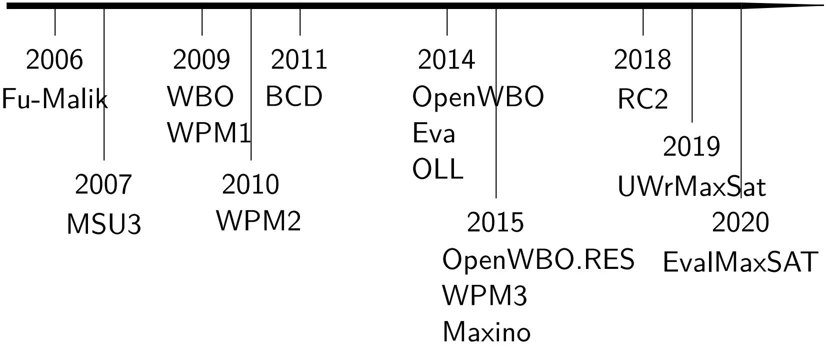



Fu-Malik [Fu and Malik, 2006]

- First core-guided algorithm for MaxSAT
- Uses multiple relaxation variables per soft clause
- Only requires AtMost1 constraints

# Core-Guided Algorithms

## Timeline

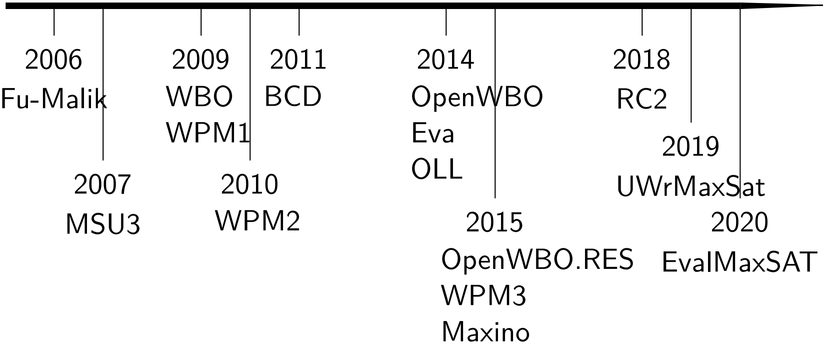



MSU3 [Marques-Silva and Planes, 2007]

- Uses one relaxation variable per soft clause
- Requires cardinality / pseudo-Boolean constraints

# Core-Guided Algorithms

## Timeline



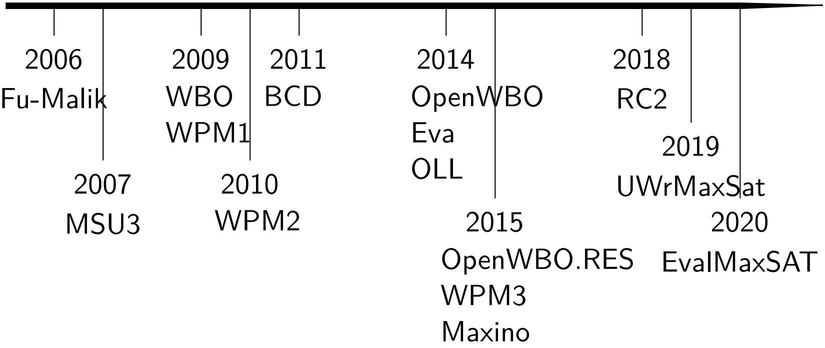

WBO [Manquinho, Marques-Silva, and Planes, 2009]

WPM1 [Ansótegui, Bonet, and Levy, 2009]

- Generalizes Fu-Malik algorithm to weighted problems
- Efficient implementation of the Fu-Malik algorithm

# Core-Guided Algorithms

## Timeline

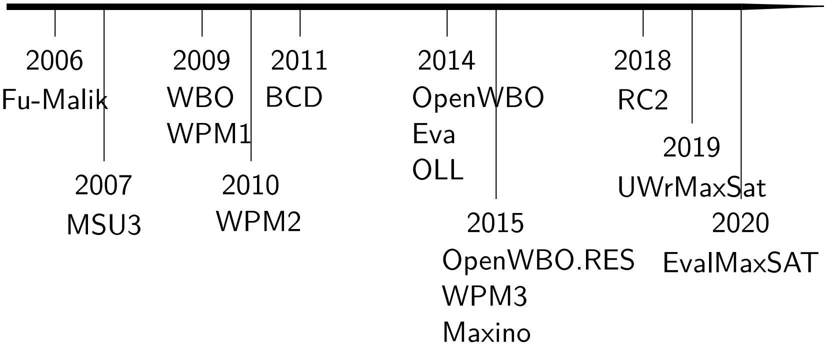



WPM2 [Ansótegui, Bonet, and Levy, 2010]

- Only one relaxation per soft clause
- Group intersecting cores into disjoint covers
- Uses a cardinality constraint per cover

# Core-Guided Algorithms

## Timeline

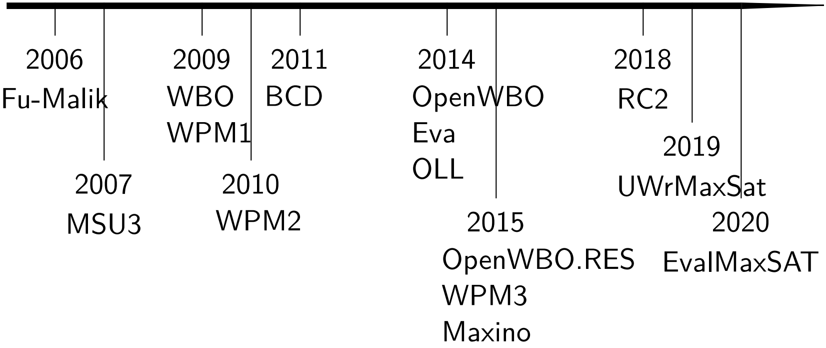



BCD [Heras, Morgado, and Marques-Silva, 2011]

- Uses binary search in core-guided algorithms

# Core-Guided Algorithms

## Timeline

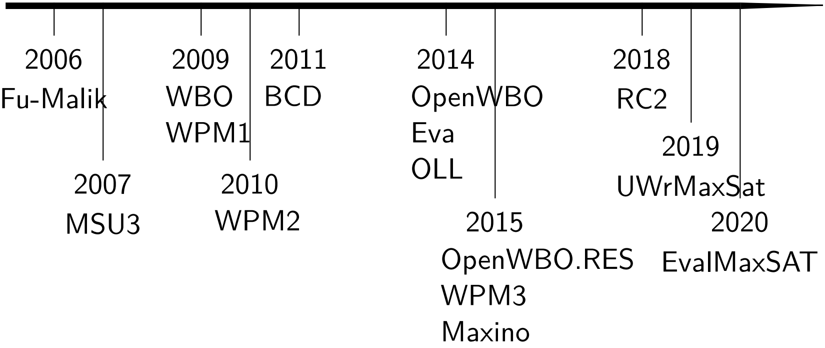



OpenWBO [Martins, Joshi, Manquinho, and Lynce, 2014a]

- Improves the MSU3 algorithm with incremental construction of cardinality constraints
- Efficient implementation of the MSU3 algorithm

# Core-Guided Algorithms

## Timeline

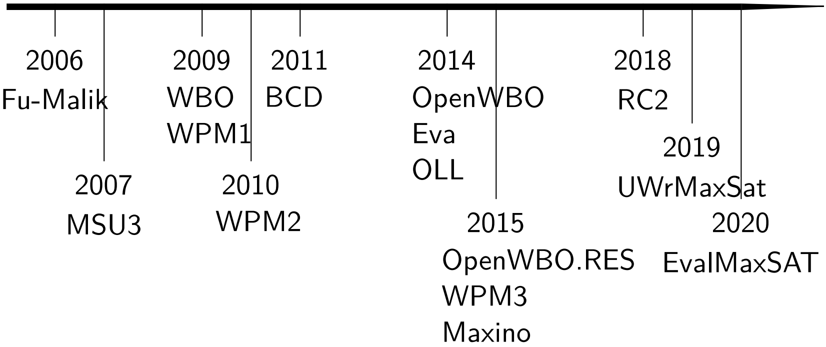



Eva [Narodytska and Bacchus, 2014]

- Uses MaxSAT resolution to refine the formula instead of using AtMost1 constraints

# Core-Guided Algorithms

## Timeline




OLL [Andres, Kaufmann, Matheis, and Schaub, 2012]

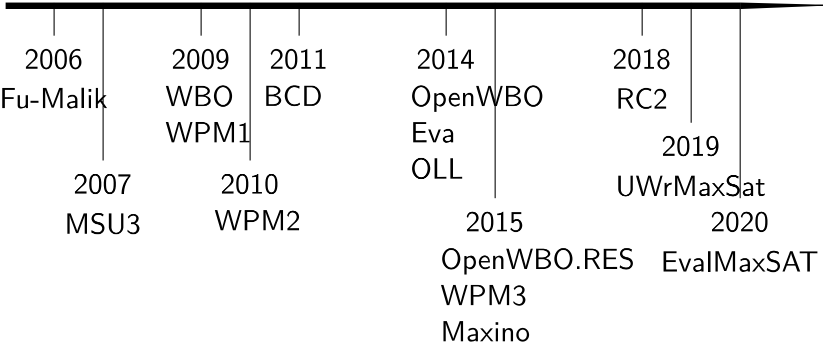
[Morgado, Dodaro, and Marques-Silva, 2014]

WPM3 [Ansótegui, Didier, and Gabàs, 2015]

- Introduce new variables to represent cardinality constraints
- $d = r_1 + r_2 + r_3 \leq 1$
- Soft clause (d, 1) is introduced

# Core-Guided Algorithms

## Timeline




OpenWBO.RES [Neves, Martins, Janota, Lynce, and Manquinho, 2015]

- Uses resolution-based graphs to partition soft clauses

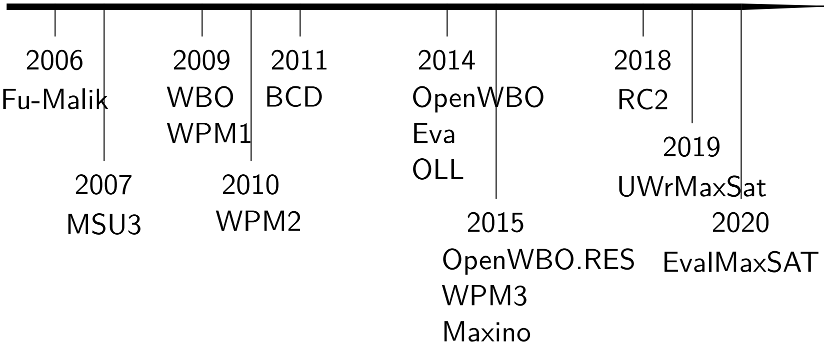
OpenWBO.RES [Neves, Martins, Janota, Lynce, and Manquinho, 2015]

Maxino [Alviano, Dodaro, and Ricca, 2015]

- Construction of the cardinality constraint uses core structure

# Core-Guided Algorithms

## Timeline




RC2 [Ignatiev, Morgado, and Marques-Silva, 2018a]

UWrMaxSat [Piotrow, 2019]

EvalMaxSAT [Avellaneda, 2020]

- Efficient implementations of the OLL algorithm
- OLL algorithm is currently the most used one

# Implicit Hitting Set Algorithms for MaxSAT

[Davies and Bacchus, 2011, 2013b,a]

# Hitting Sets and UNSAT Cores

## Hitting Sets

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

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

# Hitting Sets and UNSAT Cores

## Hitting Sets

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

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

## What does this have to do with MaxSAT?

For any MaxSAT instance $F$:
for any optimal hitting set *hs* of the *set of UNSAT cores of F*, there is an optimal solutions $\tau$ to $F$ such that $\tau$ satisfies exactly the clauses $F \setminus hs$.

# Hitting Sets and UNSAT Cores

## Key insight

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

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

# Implicit Hitting Set Approach to MaxSAT

Iterate over the following steps:

- Accumulate a collection $\mathcal{K}$ of UNSAT cores

  using a SAT solver

- Find an optimal hitting set *hs* over $\mathcal{K}$,
  and *rule out the clauses in hs for the next SAT solver call*

  *using an IP solver*

...until the SAT solver returns satisfying assignment.

# Implicit Hitting Set Approach to MaxSAT

Iterate over the following steps:

- Accumulate a collection $\mathcal{K}$ of UNSAT cores

  using a SAT solver

- Find an optimal hitting set *hs* over $\mathcal{K}$,
  and *rule out the clauses in hs for the next SAT solver call*

  *using an IP solver*

...until the SAT solver returns satisfying assignment.

## Hitting Set Problem as Integer Programming

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

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

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

# Implicit Hitting Set Approach to MaxSAT

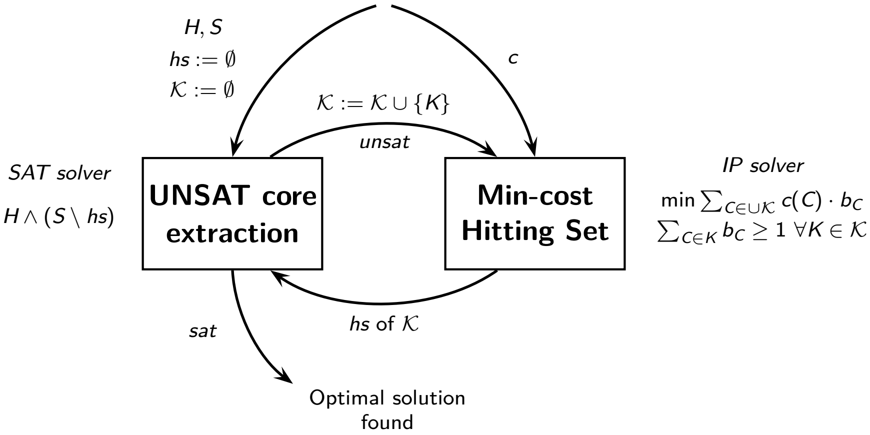
"Best out of both worlds"

Combining the main strengths of SAT and IP solvers:

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

# Solving MaxSAT by SAT and Hitting Set Computations

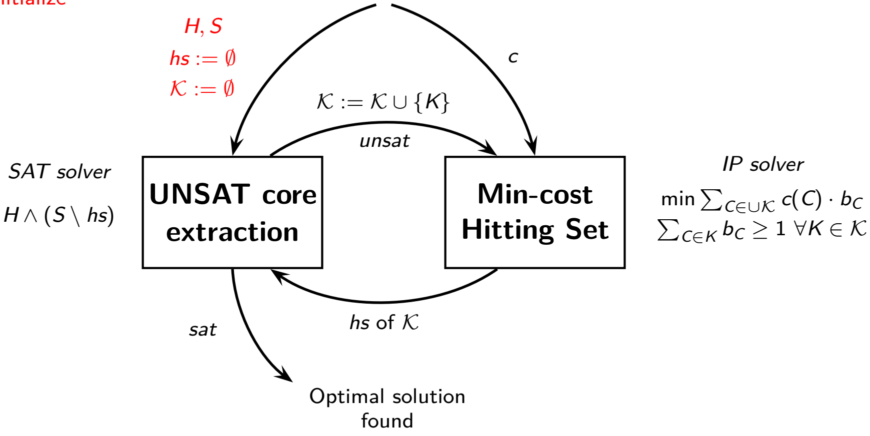
**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$



# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$

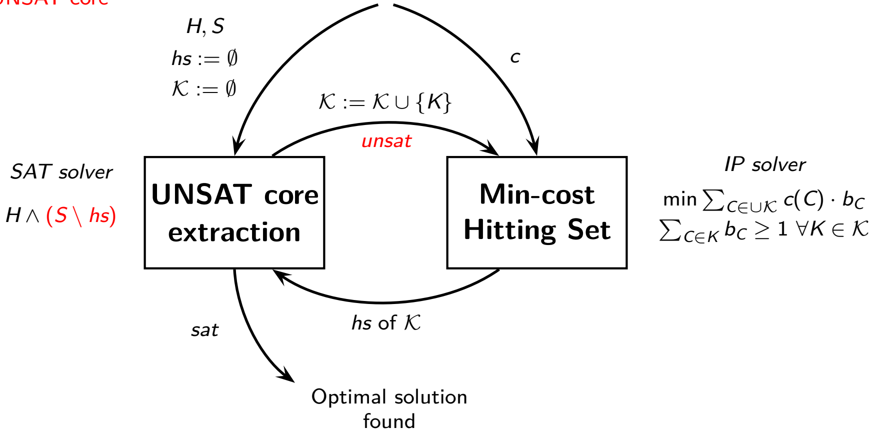
1. Initialize



# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$

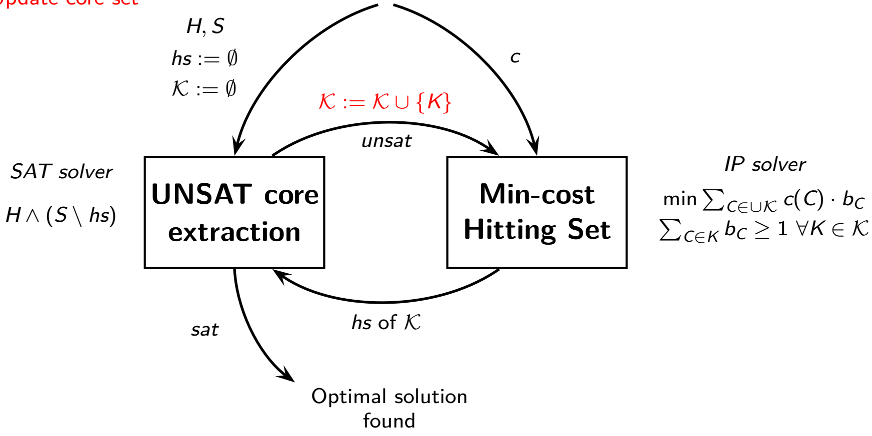
2. UNSAT core



# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$

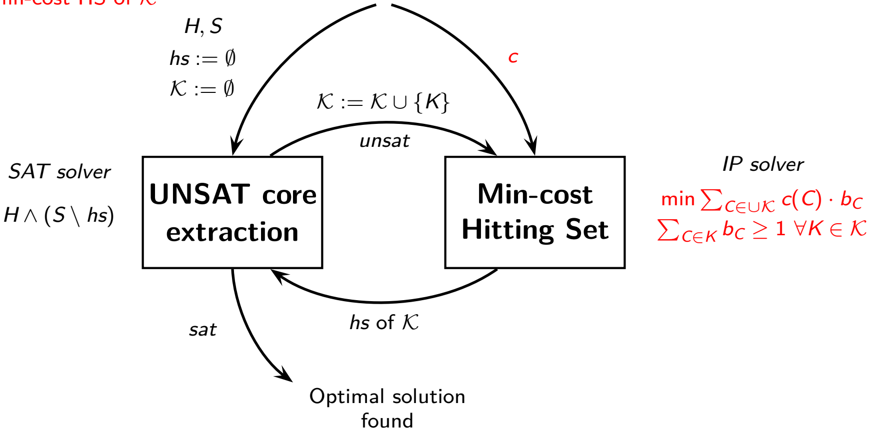
3. Update core set



# Solving MAXSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$

4. Min-cost HS of $\mathcal{K}$

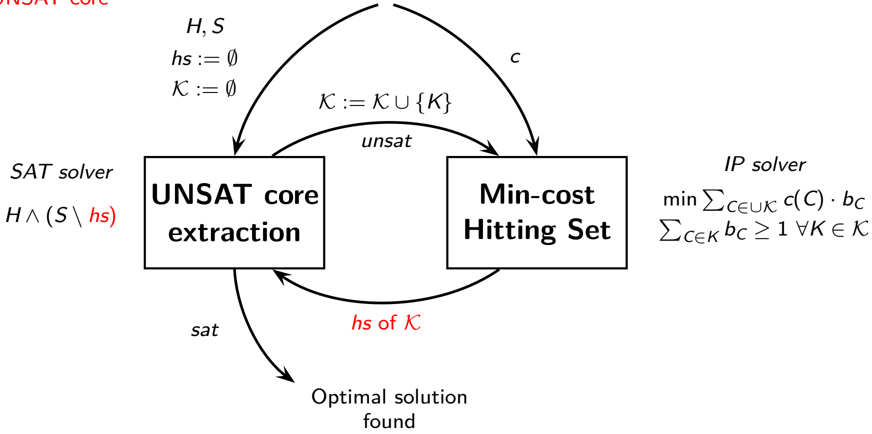


# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**

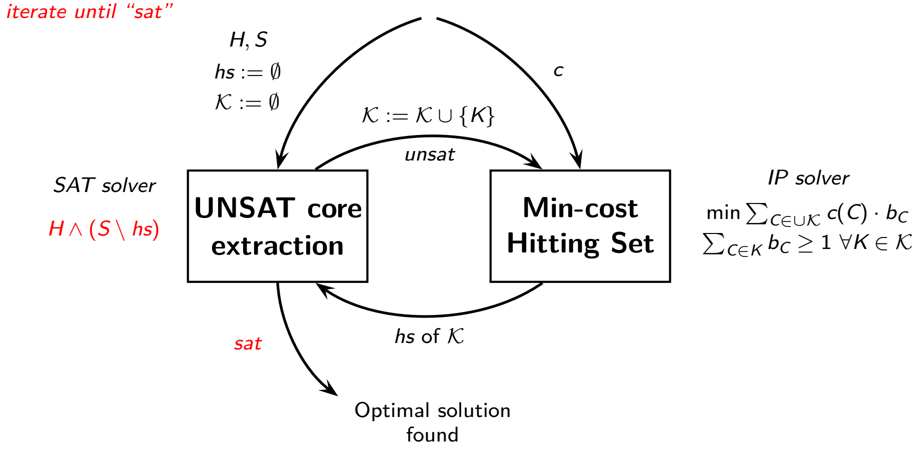
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$

5. UNSAT core



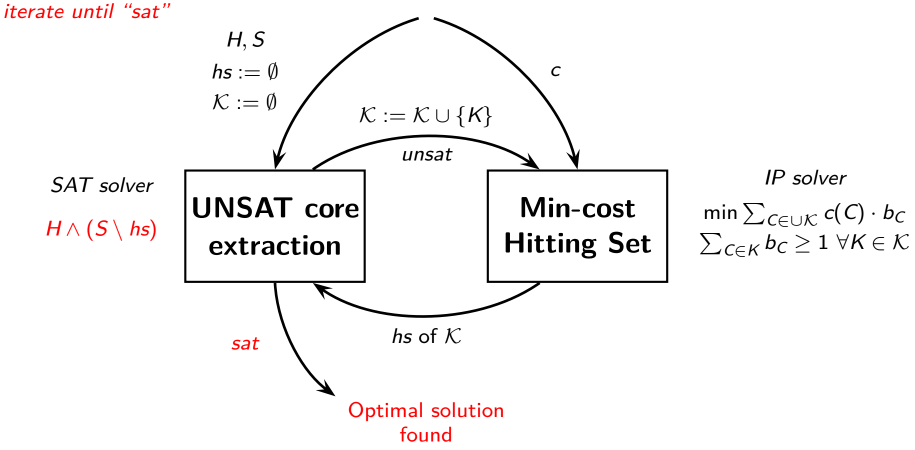
# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$



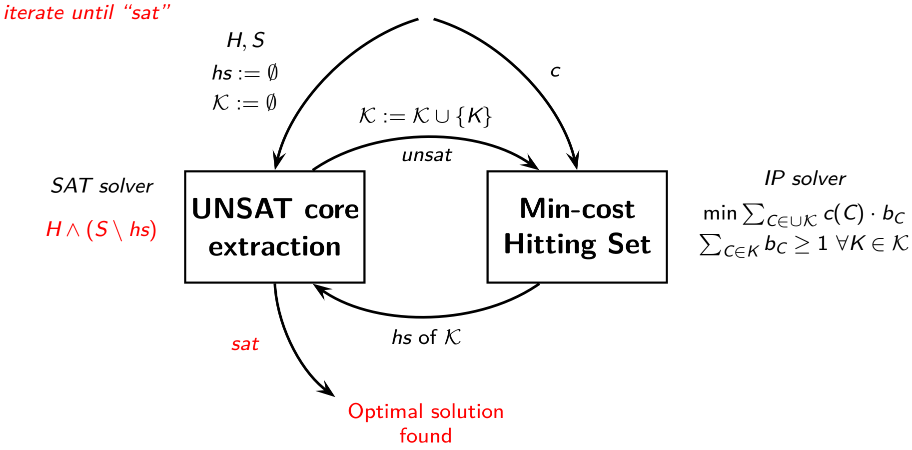
# Solving MaxSAT by SAT and Hitting Set Computations

**Input:**
hard clauses $H$, soft clauses $S$, weight function $c : S \mapsto \mathbb{R}^+$



# Solving MaxSAT by SAT and Hitting Set Computations

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



# MaxSAT by SAT and Hitting Set Computation: Example

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $H \land (S \setminus \emptyset)$

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $H \wedge (S \setminus \{C_1\})$

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $H \wedge (S \setminus \{C_1, C_9\})$

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $H \wedge (S \setminus \{C_4, C_9\})$

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

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

# MaxSAT by SAT and Hitting Set Computation: Example

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

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

- SAT solve $H \wedge (S \setminus \{C_4, C_9\}) \rightsquigarrow$ SATISFIABLE.
  Optimal cost: 2 (cost of $hs$).

# Optimizations in Solvers

Solvers implementing the implicit hittings set approach include several optimizations, such as

- a *disjoint phase* for obtaining several cores before/between hitting set computations,
  combinations of greedy and exact hitting sets computations
  [Davies and Bacchus, 2011, 2013b,a; Saikko, Berg, and Järvisalo, 2016]
- LP-solving techniques such as *reduced cost fixing*
  [Bacchus, Hyttinen, Järvisalo, and Saikko, 2017]
- abstract cores [Berg, Bacchus, and Poole, 2020]
- ...

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

# Implicit Hitting Set

- Effective on range of MaxSAT problems including large ones.
- Superior to other methods when there are many distinct weights.
- Usually superior to CPLEX.

# Incomplete MaxSAT Solving

# Why Incomplete Solving?

- Scalability
- Proving optimality often the most challenging step of complete algorithms
- Proofs of optimality not always necessary
    - Finding *good* solutions *fast*

# From Complete to Incomplete MaxSAT Solving

## Any-time algorithms

- Find intermediate (non-optimal) solutions during search.

# From Complete to Incomplete MaxSAT Solving

## Any-time algorithms

- Find intermediate (non-optimal) solutions during search.
  - Simple example: model-improving algorithm

# From Complete to Incomplete MaxSAT Solving

## Any-time algorithms

- Find intermediate (non-optimal) solutions during search.
  - Simple example: model-improving algorithm
  - *However:* also most implementations of core-guided and IHS algorithms.

# From Complete to Incomplete MaxSAT Solving

## Any-time algorithms

- Find intermediate (non-optimal) solutions during search.
  - Simple example: model-improving algorithm
  - *However:* also most implementations of core-guided and IHS algorithms.
- In other words: essentially all complete solvers can be seen as incomplete solvers.

# From Complete to Incomplete MaxSAT Solving

## Any-time algorithms

- Find intermediate (non-optimal) solutions during search.
  - Simple example: model-improving algorithm
  - *However:* also most implementations of core-guided and IHS algorithms.
- In other words: essentially all complete solvers can be seen as incomplete solvers.

## Central Question

How to combine or improve the algorithms in order to obtain good solutions faster?

# Approaches to Incomplete MaxSAT

### Model-Improving Incomplete Search

How to improve the model-improving algorithm for incomplete search.

*complete & any-time*

### Stochastic Local Search (SLS)

Quickly traverse the search space by local changes to current solution *incomplete*

### Core-Boosted search

Combine core-guided and model-improving search.

*complete & any-time*

### SLS with a SAT solver

Local search over which soft clauses should be satisfied, check with a SAT solver. *incomplete*

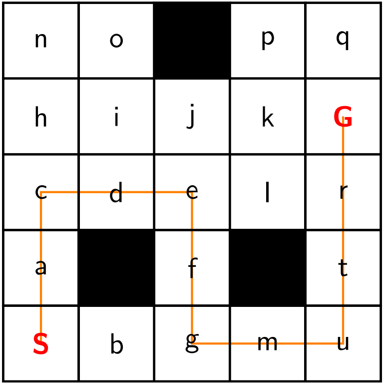
# Model-Improving Algorithm for Incomplete Solving

# Recall

## Model-Improving Algorithm

### Intuition

Improve a best known solution with a SAT solver until no better ones can be found.




$\mathrm{UB} = 10$

$\mathrm{SAT\text{-}SOLVE}\,(H \wedge \textsc{CostLessThan}(S, UB))$

$\tau^1 = \{S, a, c, d, e, f, g, m, u, t, r, G, \neg b, \neg l, \ldots, \neg q\}$

$cost(\tau^1) = 10$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.

Size depends on:
number of soft clauses,
diversity of weights, and UB

$$\text{SAT-SOLVE}\,(H \wedge \textsc{CostLessThan}(S, UB))$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \rightarrow (a + b = 1)), \\ (a \rightarrow (c + S = 2)), \ldots, \\ (b \rightarrow (S + g = 2)), \ldots, \\ (g \rightarrow (b + f + m = 2)), \ldots, \\ (e \rightarrow (j + d + l + f = 2)), \ldots, \\ (G \rightarrow (q + k + r))\} \end{array}$$

$$S = \begin{array}{l} \{(\neg a), (\neg b), (\neg c), (\neg d), \\ (\neg e), (\neg f), (\neg g), (\neg h), \\ (\neg i), (\neg j), (\neg k), (\neg l), \\ (\neg m), (\neg n), (\neg o)(\neg p), \\ (\neg q), (\neg r), (\neg t), (\neg u)\} \end{array}$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \rightarrow (a + b = 1)), \\ (a \rightarrow (c + S = 2)), \ldots, \\ (b \rightarrow (S + g = 2)), \ldots, \\ (g \rightarrow (b + f + m = 2)), \ldots, \\ (e \rightarrow (j + d + l + f = 2)), \ldots, \\ (G \rightarrow (q + k + r))\} \end{array}$$

$$\begin{aligned} S^1 &= \{(\neg a), (\neg b), (\neg c), (\neg d)\} \\ S^2 &= \{(\neg e), (\neg f), (\neg g), (\neg h)\} \\ S^3 &= \{(\neg i), (\neg j), (\neg k), (\neg l)\} \\ S^4 &= \{(\neg m), (\neg n), (\neg o)(\neg p)\} \\ S^5 &= \{(\neg q), (\neg r), (\neg t), (\neg u)\} \end{aligned}$$

$$\text{MODEL-IMPROVE}\left(H, S^1\right)$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \rightarrow (a + b = 1)), \\ (a \rightarrow (c + S = 2)), \ldots, \\ (b \rightarrow (S + g = 2)), \ldots, \\ (g \rightarrow (b + f + m = 2)), \ldots, \\ (e \rightarrow (j + d + l + f = 2)), \ldots, \\ (G \rightarrow (q + k + r))\} \end{array}$$

$$\begin{aligned} S^1 &= \{(\neg a), (\neg b), (\neg c), (\neg d)\} \\ S^2 &= \{(\neg e), (\neg f), (\neg g), (\neg h)\} \\ S^3 &= \{(\neg i), (\neg j), (\neg k), (\neg l)\} \\ S^4 &= \{(\neg m), (\neg n), (\neg o)(\neg p)\} \\ S^5 &= \{(\neg q), (\neg r), (\neg t), (\neg u)\} \end{aligned}$$

$$\text{MODEL-IMPROVE}\left(H, S^1 \cup S^2\right)$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \to (a + b = 1)), \\ (a \to (c + S = 2)), \dots, \\ (b \to (S + g = 2)), \dots, \\ (g \to (b + f + m = 2)), \dots, \\ (e \to (j + d + l + f = 2)), \dots, \\ (G \to (q + k + r))\} \end{array}$$

$$\begin{array}{l} S^1 = \{(\neg a), (\neg b), (\neg c), (\neg d)\} \\ S^2 = \{(\neg e), (\neg f), (\neg g), (\neg h)\} \\ S^3 = \{(\neg i), (\neg j), (\neg k), (\neg l)\} \\ S^4 = \{(\neg m), (\neg n), (\neg o)(\neg p)\} \\ S^5 = \{(\neg q), (\neg r), (\neg t), (\neg u)\} \end{array}$$

$$\text{MODEL-IMPROVE}\left(H, S^1 \cup S^2 \cup S^3\right)$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \rightarrow (a + b = 1)), \\ (a \rightarrow (c + S = 2)), \ldots, \\ (b \rightarrow (S + g = 2)), \ldots, \\ (g \rightarrow (b + f + m = 2)), \ldots, \\ (e \rightarrow (j + d + l + f = 2)), \ldots, \\ (G \rightarrow (q + k + r))\} \end{array}$$

$$S^1 = \{(\neg a), (\neg b), (\neg c), (\neg d)\}$$
$$S^2 = \{(\neg e), (\neg f), (\neg g), (\neg h)\}$$
$$S^3 = \{(\neg i), (\neg j), (\neg k), (\neg l)\}$$
$$S^4 = \{(\neg m), (\neg n), (\neg o)(\neg p)\}$$
$$S^5 = \{(\neg q), (\neg r), (\neg t), (\neg u)\}$$

$$\text{MODEL-IMPROVE}\left(H, S^1 \cup S^2 \cup S^3 \cup S^4\right)$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses

$$H = \begin{array}{l} \{(S), (G), (S \rightarrow (a + b = 1)), \\ (a \rightarrow (c + S = 2)), \ldots, \\ (b \rightarrow (S + g = 2)), \ldots, \\ (g \rightarrow (b + f + m = 2)), \ldots, \\ (e \rightarrow (j + d + l + f = 2)), \ldots, \\ (G \rightarrow (q + k + r))\} \end{array}$$

$$S^1 = \{(\neg a), (\neg b), (\neg c), (\neg d)\}$$
$$S^2 = \{(\neg e), (\neg f), (\neg g), (\neg h)\}$$
$$S^3 = \{(\neg i), (\neg j), (\neg k), (\neg l)\}$$
$$S^4 = \{(\neg m), (\neg n), (\neg o)(\neg p)\}$$
$$S^5 = \{(\neg q), (\neg r), (\neg t), (\neg u)\}$$

$$\text{MODEL-IMPROVE}\left(H, S^1 \cup S^2 \cup S^3 \cup S^4 \cup S^5\right)$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of CostLessThan($S, UB$) can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses
  - Rescale weights.

$$S = \{(C_1, 100), (C_2, 1200), (C_3, 1540) \ldots\}$$

Divide by 100

$$S = \{(C_1, 1), (C_2, 12), (C_3, 15) \ldots\}$$

# Model-Improving Incomplete Search

Joshi et al. [2018]; Demirovic and Stuckey [2019]

## Key Challenges

- Encoding of $\textsc{CostLessThan}(S, UB)$ can be (and often is) large
  - Especially with weights.
- Proposed Improvements:
  - Partition soft clauses
  - Rescale weights.
  - Core-Boosted Search (more on this later)

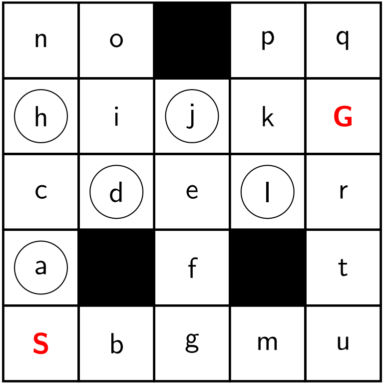
# Stochastic Local Search for Incomplete MaxSAT

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.




$\mathrm{UB} = \infty$

$\tau^{\text{BEST}} = \emptyset$

$$\tau^{\text{CUR}} = \{S, a, d, h, j, l, G, \neg b, \neg c, \neg e, \neg f, \neg g, \ldots, \neg q\}$$

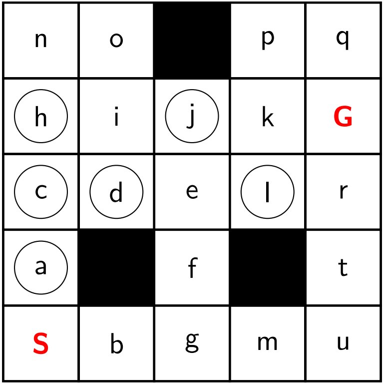
Not a solution: $\tau^{\text{CUR}}(a \to (S + c = 2)) = 0$

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.
2. Iteratively flip literals.




$\mathrm{UB} = \infty$

$\tau^{\mathrm{BEST}} = \emptyset$

*Flip value of: c*

$$\tau^{\mathrm{CUR}} = \quad \{S, a, \mathbf{c}, d, h, j, l, G, \neg b, \neg e, \neg f, \neg g, \ldots, \neg q\}$$

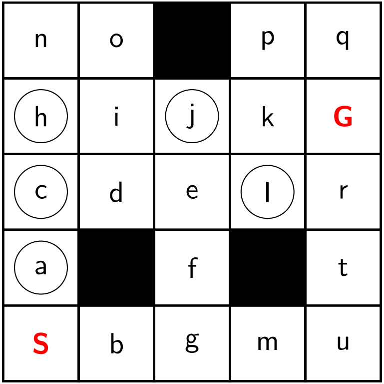
Not a solution: $\tau^{\mathrm{CUR}}(c \rightarrow (h + d + a = 2)) = 0$

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.
2. Iteratively flip literals.




$\mathrm{UB} = \infty$

$\tau^{\text{BEST}} = \emptyset$

*Flip value of: d*

$\tau^{\text{CUR}} = \{S, a, c, h, j, l, G, \neg b, \neg \mathbf{d}, \neg e, \neg f, \neg g, \ldots, \neg q\}$

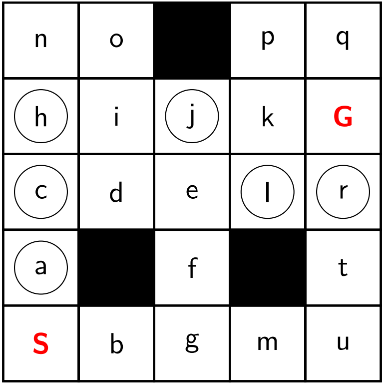
Not a solution: $\tau^{\text{CUR}}(G \rightarrow (k + r = 1)) = 0$

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.
2. Iteratively flip literals.




$\mathrm{UB} = \infty$

$\tau^{\text{BEST}} = \emptyset$

*Flip value of:* $r$

$\tau^{\text{CUR}} = \{S, a, c, h, j, l, \mathbf{r}, G, \neg b, \neg d, \neg e, \neg f, \neg g, \ldots, \neg q\}$

Not a solution: $\tau^{\text{CUR}}(l \rightarrow (e + k + r = 2)) = 0$

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.
2. Iteratively flip literals.

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

$\mathrm{UB} = \infty$

$\tau^{\text{BEST}} = \emptyset$

*Flip value of: e*

$\tau^{\text{CUR}} = \{S, a, c, h, j, \mathbf{e}, l, r, G, \neg b, \neg d, \neg f, \neg g, \ldots, \neg q\}$

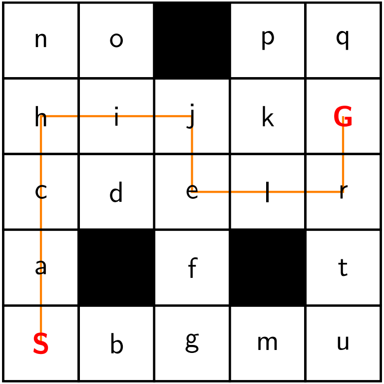
Not a solution: $\tau^{\text{CUR}}(h \rightarrow (n + i + c = 2)) = 0$

# SLS for Incomplete MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Intuition

1. Initialise a random assignment.
2. Iteratively flip literals.
3. Check cost of any solutions and update UB when needed.




$\mathrm{UB} = \mathbf{8}$

$\tau^{\text{BEST}} = \tau^{\text{CUR}}$

*Flip value of: i*

$$\tau^{\text{CUR}} = \{S, a, c, h, j, e, \mathbf{i}, l, r, G, \neg b, \neg d, \neg e, \neg f, \neg g, \ldots, \neg q\}$$

Is a solution: $cost(\tau^{\text{CUR}}) = 8$

# SLS for MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Key challenges

- How to guarantee that solutions satisfy hard clauses?
- How to make use of the weights?

# SLS for MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Key challenges

- How to guarantee that solutions satisfy hard clauses?
- How to make use of the weights?

## Proposed solutions:

# SLS for MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Key challenges

- How to guarantee that solutions satisfy hard clauses?
- How to make use of the weights?

## Proposed solutions:

- Extend weights to all clauses
  - Initialize weight of all hard clauses to 1

# SLS for MaxSAT

Cai et al. [2016]; Luo et al. [2017]

## Key challenges

- How to guarantee that solutions satisfy hard clauses?
- How to make use of the weights?

## Proposed solutions:

- Extend weights to all clauses
  - Initialize weight of all hard clauses to 1
- Flip literals from unsatisfied clauses with high weight.
- Periodically increase weights of clauses that are frequently unsatisfied.

# Core-Boosted Search for Incomplete MaxSAT

# Core-Boosted Search - Intuition

Berg et al. [2019]

## Recall - Core-Guided search

- Extract a core $K$
- Relax the instance s.t. one clause from $K$ can be unsatisfied in future iterations
- Continue until no more cores can be found.

# Core-Boosted Search - Intuition

Berg et al. [2019]

## Recall - Core-Guided search

- Extract a core $K$
- Relax the instance s.t. one clause from $K$ can be unsatisfied in future iterations
- Continue until no more cores can be found.

## Alternative view

- Any solution to $F$ falsifies at least one clause in $K$
  - $K$ proves an additional LB of 1 on $cost(F)$.
- "Relaxing the instance" $\rightarrow$ "Lowering $cost(F)$ by 1"

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.

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

Instance $F$

Solutions correspond to paths between $S$ and $G$

$cost(F) = 6$ i.e

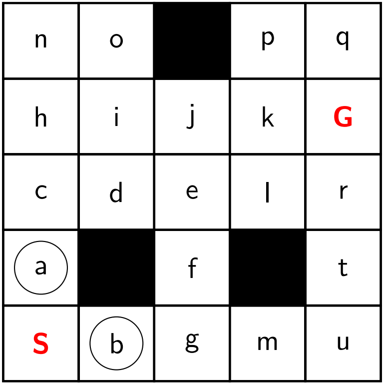
Length of shortest path from $S$ to $G$

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.




Instance $F$

Solutions correspond to paths between $S$ and $G$

$cost(F) = 6$ i.e

Length of shortest path from $S$ to $G$

Core: $\{(\neg a), (\neg b)\}$

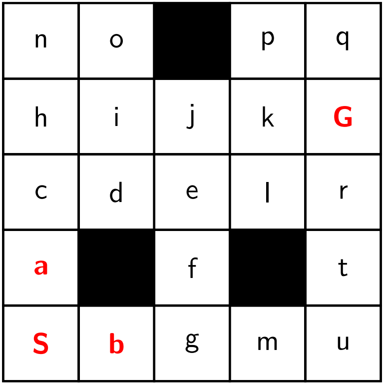
i.e. all paths go through $a$ or $b$

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.




Instance

$\mathrm{REFORM}(F, \{(\neg a), (\neg b)\})$

Solutions correspond to paths between $S$ and $G$

$cost(F) = 5$ i.e

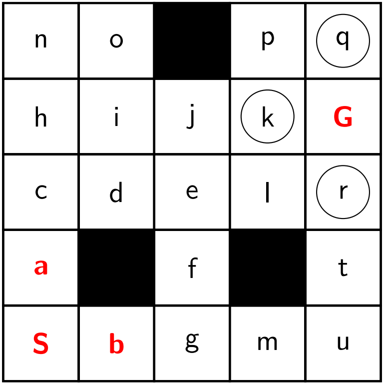
Length of shortest path from either $a$ or $b$ to $G$

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.




Instance

$\mathrm{REFORM}(F, \{(\neg a), (\neg b)\})$

Solutions correspond to paths between $S$ and $G$

$cost(F) = 5$ i.e

Length of shortest path from either $a$ or $b$ to $G$

Core: $\{(\neg q), (\neg k), (\neg r)\}$

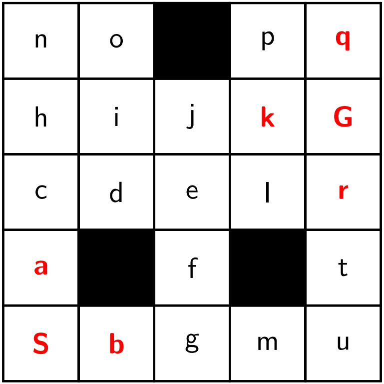
i.e. all paths go through $q$, $k$ or $r$

# Core-Boosted Search

## Example

### Intuition

Cores prove that all paths go through specific nodes.
Reformulating restricts search to paths between the remaining nodes.




Instance

$\mathrm{REFORM}(F, \{(\neg a), (\neg b)\}, \{(\neg q), (\neg k), (\neg r)\})$

Solutions correspond to paths between $S$ and $G$

$cost(F) = 4$ i.e

Length of shortest path from either $a$ or $b$ to either $q$, $k$ or $r$

# Core-Boosted Linear Search

## In General

Solving: $F$

# Core-Boosted Linear Search

## In General

Solving: $F$

*Input*

$\boxed{\textsc{Core-Guided}(F)}$

# Core-Boosted Linear Search

## In General

Solving: $F$

*Input*

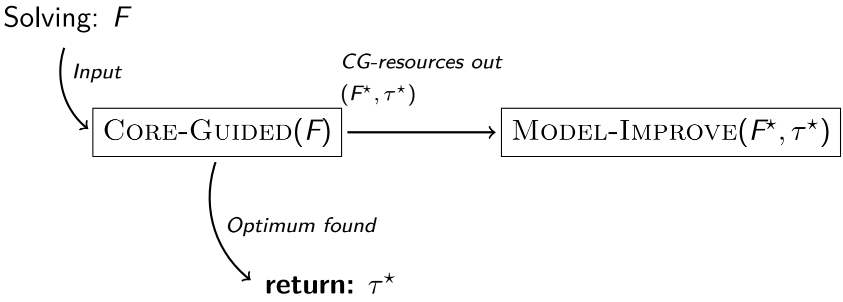
$\boxed{\textsc{Core-Guided}(F)}$

*Optimum found*

**return:** $\tau^{\star}$

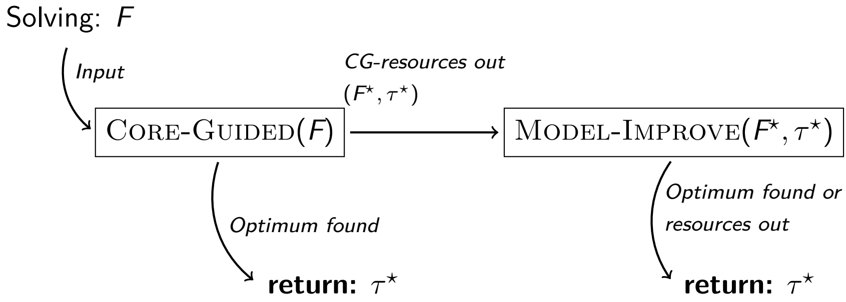
# Core-Boosted Linear Search

## In General



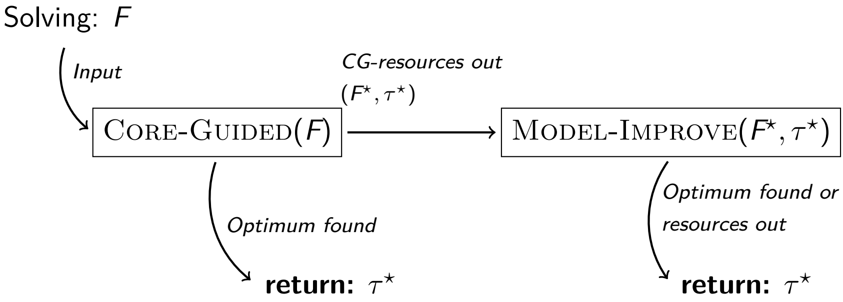
# Core-Boosted Linear Search

## In General



# Core-Boosted Linear Search

## In General




Further improvements by including SLS prior to core-guided phase.
**State-of-the-art performance (in 2020) on unweighted instances**.

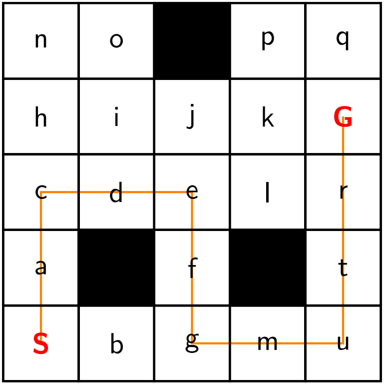
# Local Search with a SAT Solver

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$




$S = \{(\neg a), (\neg b), (\neg c), \ldots\}$

$\mathrm{UB} = 10$

$\tau^* = \{S, a, c, \ldots, G, \neg b, \neg l, \ldots, \neg q\}$

$cost(\tau^*) = 10$

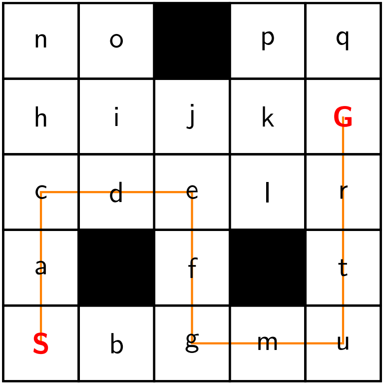
$\mathrm{FIXED} = \emptyset$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{a}), (\neg b), (\neg c), \ldots\}$

$\mathrm{UB} = 10$

$\tau^* = \{S, a, c, \ldots, G, \neg b, \neg l, \ldots, \neg q\}$

$cost(\tau^*) = 10$

$\mathrm{FIXED} = \emptyset$

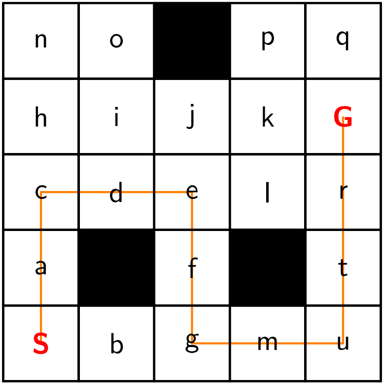
Current: $(\neg a)$ $\tau^*(\neg a) = 0$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{a}), (\neg b), (\neg c), \ldots\}$

$\mathrm{UB} = 10$

$\tau^* = \{S, a, c, \ldots, G, \neg b, \neg l, \ldots, \neg q\}$

$cost(\tau^*) = 10$

$\mathrm{FIXED} = \emptyset$

Current: $(\neg a)$ $\tau^*(\neg a) = 0$

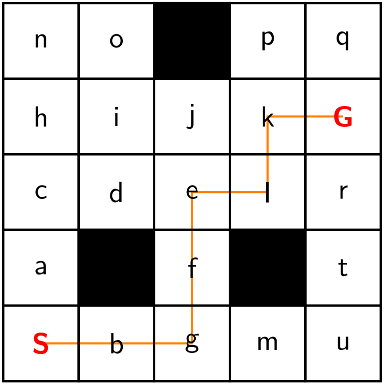
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg a))$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{a}), (\neg b), (\neg c), \ldots\}$

$\mathrm{UB} = \mathbf{6}$

$\tau^* = \{\mathbf{S}, \mathbf{b}, \mathbf{g}, \mathbf{f}, \mathbf{e}, \mathbf{l}, \mathbf{k}, \mathbf{G}, \neg\mathbf{a}, \neg\mathbf{c}, \neg\mathbf{d}, \ldots, \neg\mathbf{q}\}$

$cost(\tau^*) = \mathbf{6}$

$\mathrm{FIXED} = \{(\neg \mathbf{a})\}$

Current: $(\neg a)$ $\tau^*(\neg a) = 0$

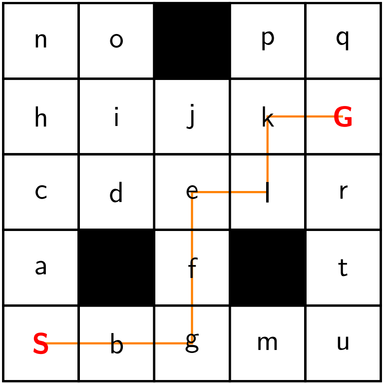
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg a))$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{b}), (\neg c), (\neg d), \ldots\}$

$\mathrm{UB} = 6$

$\tau^* = \{S, b, g, f, e, l, k, G, \neg a, \neg c, \neg d, \ldots, \neg q\}$

$cost(\tau^*) = 6$

$\mathrm{FIXED} = \{(\neg a)\}$

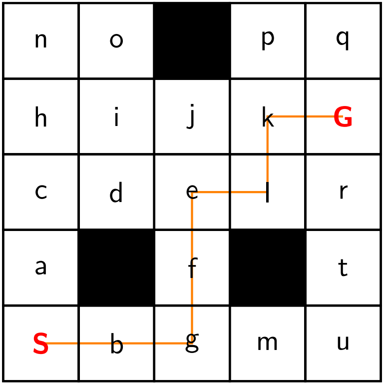
Current: $(\neg b)$ $\tau^*(\neg b) = 0$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{b}), (\neg c), (\neg d), \ldots\}$

$\mathrm{UB} = 6$

$\tau^* = \{S, b, g, f, e, l, k, G, \neg a, \neg c, \neg d, \ldots, \neg q\}$

$cost(\tau^*) = 6$

$\mathrm{FIXED} = \{(\neg a)\}$

Current: $(\neg b)$ $\tau^*(\neg b) = 0$

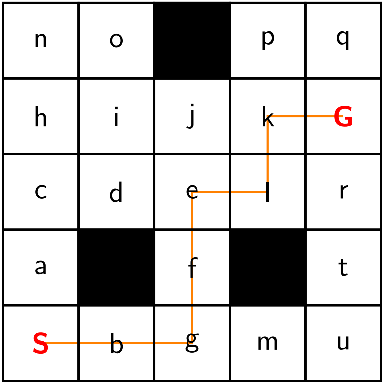
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg b))$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{b}), (\neg c), (\neg d), \ldots\}$

$\mathrm{UB} = 6$

$\tau^* = \{S, b, g, f, e, l, k, G, \neg a, \neg c, \neg d, \ldots, \neg q\}$

$cost(\tau^*) = 6$

$\mathrm{FIXED} = \{(\neg \mathbf{a}), (\mathbf{b})\}$

Current: $(\neg b)$ $\tau^*(\neg b) = 0$

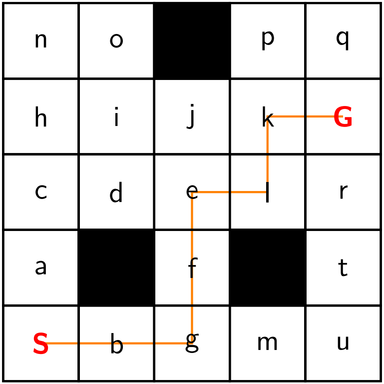
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg b))$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{c}), (\neg d), (\neg e), \ldots\}$

$\mathrm{UB} = 6$

$\tau^* = \{S, b, g, f, e, l, k, G, \neg a, \neg c, \neg d, \ldots, \neg q\}$

$cost(\tau^*) = 6$

$\mathrm{FIXED} = \{(\neg a), (b)\}$

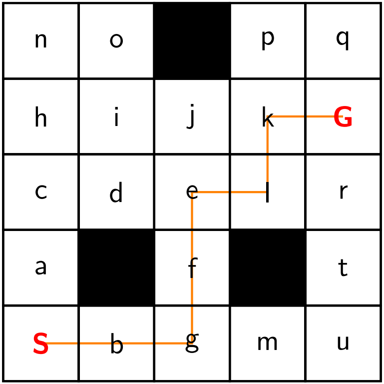
Current: $(\neg c)$ $\qquad$ $\tau^*(\neg c) = 0$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.




$S = \{(\neg \mathbf{c}), (\neg d), (\neg e), \ldots\}$

$\mathrm{UB} = 6$

$\tau^* = \{S, b, g, f, e, l, k, G, \neg a, \neg c, \neg d, \ldots, \neg q\}$

$cost(\tau^*) = 6$

$\mathrm{FIXED} = \{(\neg \mathbf{a}), (\mathbf{b}), (\neg \mathbf{c})\}$

Current: $(\neg c)$ $\tau^*(\neg c) = 0$

# SAT-based SLS

Nadel [2018, 2019]

## Intuition

1. Obtain any solution $\tau^*$
2. Improve $\tau^*$ by enforcing the satisfaction of an increasing subset of soft clauses.

**Further improvements by more sophisticated ways of ordering soft clauses**
**State-of-the-art performance on weighted instances**

# (Some of the) solvers in the latest evaluation

| Solver | SLS | Model Improving | Core-Guided | SAT-based SLS | Other |
|---|---|---|---|---|---|
| Loandra | | x | x | | |
| StableResolver | x | | | | x |
| TT-Open-WBO-Inc | | | | x | |
| sls-mcs | x | | x | | |
| sls-lsu | | | | | |
| SATLike-c | x | x | x | | |
| Open-WBO-Inc-complete | | x | x | | x |
| Open-WBO-Inc-satlike | x | | x | x | |

# (Some of the) solvers in the latest evaluation

| Solver | SLS | Model Improving | Core-Guided | SAT-based SLS | Other |
|---|---|---|---|---|---|
| Loandra | | x | x | | |
| StableResolver | x | | | | x |
| TT-Open-WBO-Inc | | | | x | |
| sls-mcs | x | | x | | |
| sls-lsu | | | | | |
| SATLike-c | x | x | x | | |
| Open-WBO-Inc-complete | | x | x | | x |
| Open-WBO-Inc-satlike | x | | x | x | |

**Take Home Message**

Effective incomplete solvers make use of several different algorithms.

# Incomplete MaxSAT

## Summary

- Incomplete MaxSAT solving seeks to address scalability without sacrificing solution quality (too much)

# Incomplete MaxSAT

## Summary

- Incomplete MaxSAT solving seeks to address scalability without sacrificing solution quality (too much)
- Several different approaches developed in recent years

# Incomplete MaxSAT

## Summary

- Incomplete MaxSAT solving seeks to address scalability without sacrificing solution quality (too much)
- Several different approaches developed in recent years
  - Orthogonal performance on different domains.
  - Best solvers combine several different algorithms

# Incomplete MaxSAT

## Summary

- Incomplete MaxSAT solving seeks to address scalability without sacrificing solution quality (too much)
- Several different approaches developed in recent years
  - Orthogonal performance on different domains.
  - Best solvers combine several different algorithms

### Take Home Message - Which solver to choose?

Short answer: Depends on the domain.

# Incomplete MaxSAT

## Summary

- Incomplete MaxSAT solving seeks to address scalability without sacrificing solution quality (too much)
- Several different approaches developed in recent years
  - Orthogonal performance on different domains.
  - Best solvers combine several different algorithms

### Take Home Message - Which solver to choose?

Short answer: Depends on the domain.
Longer answer (in 2020): Try TT-Open-WBO-Inc for weighted and SATLike (2020 version) or Loandra for unweighted.

# Real-World Applications of MaxSAT

# Overview of Applications

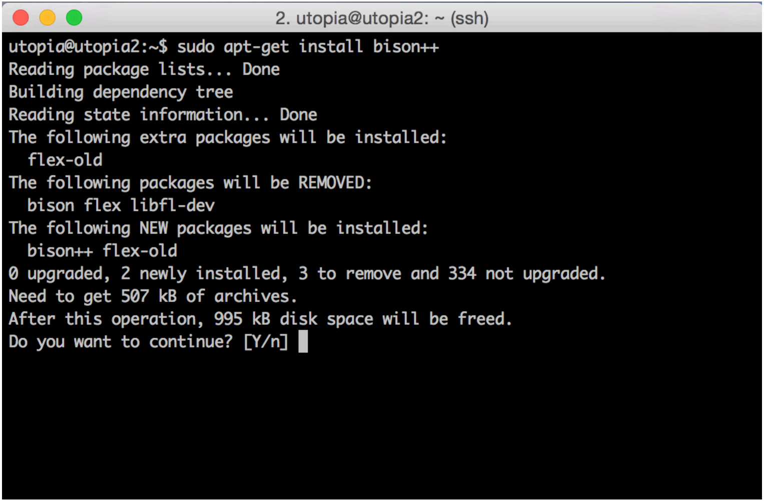
# Overview of Applications


Examples of real-world applications:

- Package upgradeability
- Learning interpretable classification rules

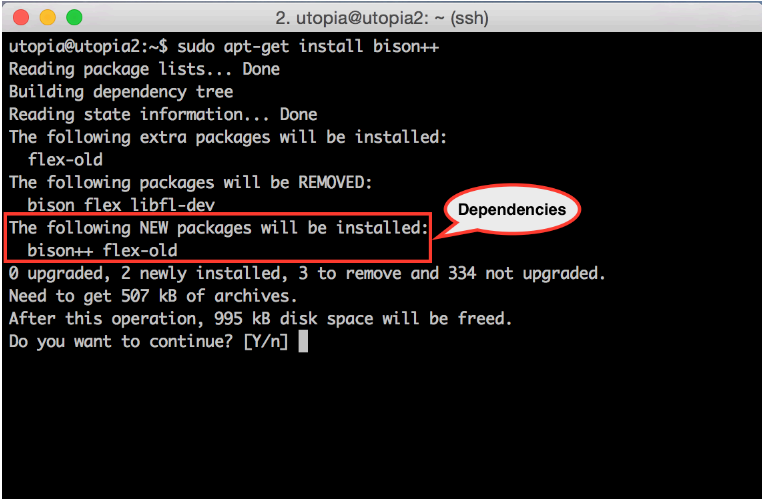
# Real-World Applications of MaxSAT: Package Upgradeability

# Software Package Upgradeability Problem

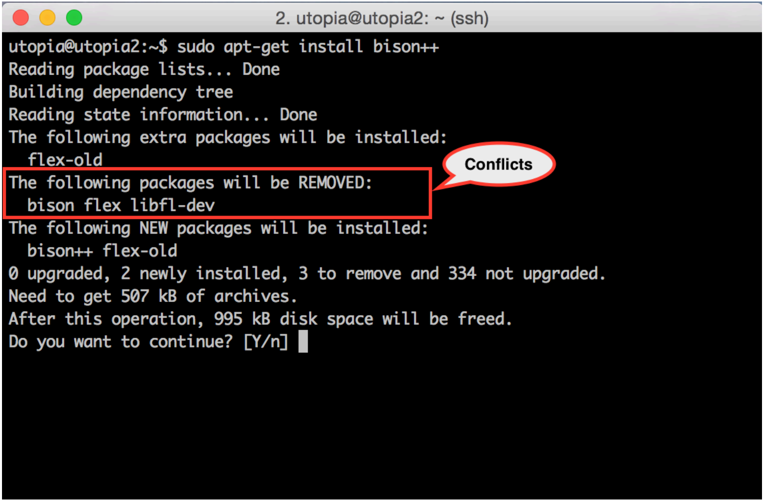



```
utopia@utopia2:~$ sudo apt-get install bison++
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  flex-old
The following packages will be REMOVED:
  bison flex libfl-dev
The following NEW packages will be installed:
  bison++ flex-old
0 upgraded, 2 newly installed, 3 to remove and 334 not upgraded.
Need to get 507 kB of archives.
After this operation, 995 kB disk space will be freed.
Do you want to continue? [Y/n]
```

# Software Package Upgradeability Problem



# Software Package Upgradeability Problem



# Software Package Upgradeability Problem

| Package | Dependencies | Conflicts |
|---|---|---|
| $p_1$ | $\{p_2 \vee p_3\}$ | $\{p_4\}$ |
| $p_2$ | $\{p_3\}$ | $\{\}$ |
| $p_3$ | $\{p_2\}$ | $\{p_4\}$ |
| $p_4$ | $\{p_2 \wedge p_3\}$ | $\{\}$ |

- Set of packages we want to install: $\{p_1, p_2, p_3, p_4\}$
- Each package $p_i$ has a set of **dependencies**:
  - Packages that must be installed for $p_i$ to be installed
- Each package $p_i$ has a set of **conflicts**:
  - Packages that cannot be installed for $p_i$ to be installed

# Software Package Upgradeability Problem as SAT

| Package | Dependencies | Conflicts |
|---|---|---|
| $p_1$ | $\{p_2 \vee p_3\}$ | $\{p_4\}$ |
| $p_2$ | $\{p_3\}$ | $\{\}$ |
| $p_3$ | $\{p_2\}$ | $\{p_4\}$ |
| $p_4$ | $\{p_2 \wedge p_3\}$ | $\{\}$ |

How can we encode this problem to Boolean Satisfiability?

# Software Package Upgradeability Problem as SAT

| Package | Dependencies | Conflicts |
|---|---|---|
| $p_1$ | $\{p_2 \vee p_3\}$ | $\{p_4\}$ |
| $p_2$ | $\{p_3\}$ | $\{\}$ |
| $p_3$ | $\{p_2\}$ | $\{p_4\}$ |
| $p_4$ | $\{p_2 \wedge p_3\}$ | $\{\}$ |

How can we encode this problem to Boolean Satisfiability?

- Encoding dependencies:
  - $p_1 \Rightarrow (p_2 \vee p_3) \equiv (\neg p_1 \vee p_2 \vee p_3)$
  - $p_2 \Rightarrow p_3 \equiv (\neg p_2 \vee p_3)$
  - $p_3 \Rightarrow p_2 \equiv (\neg p_3 \vee p_2)$
  - $p_4 \Rightarrow (p_2 \wedge p_3) \equiv (\neg p_4 \vee p_2) \wedge (\neg p_4 \vee p_3)$

# Software Package Upgradeability Problem as SAT

| Package | Dependencies | Conflicts |
|---|---|---|
| $p_1$ | $\{p_2 \vee p_3\}$ | $\{p_4\}$ |
| $p_2$ | $\{p_3\}$ | $\{\}$ |
| $p_3$ | $\{p_2\}$ | $\{p_4\}$ |
| $p_4$ | $\{p_2 \wedge p_3\}$ | $\{\}$ |

How can we encode this problem to Boolean Satisfiability?

- Encoding conflicts:
  - $p_1 \Rightarrow \neg p_4 \equiv (\neg p_1 \vee \neg p_4)$
  - $p_3 \Rightarrow \neg p_4 \equiv (\neg p_3 \vee \neg p_4)$

# Software Package Upgradeability Problem as SAT

| Package | Dependencies | Conflicts |
|---|---|---|
| $p_1$ | $\{p_2 \vee p_3\}$ | $\{p_4\}$ |
| $p_2$ | $\{p_3\}$ | $\{\}$ |
| $p_3$ | $\{p_2\}$ | $\{p_4\}$ |
| $p_4$ | $\{p_2 \wedge p_3\}$ | $\{\}$ |

How can we encode this problem to Boolean Satisfiability?

- Encoding installing all packages:
  - $(p_1) \wedge (p_2) \wedge (p_3) \wedge (p_4)$

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

Dependencies $\quad \neg p_1 \vee p_2 \vee p_3 \quad \neg p_2 \vee p_3 \quad \neg p_3 \vee p_2$

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

| | | | | |
|---|---|---|---|---|
| Dependencies | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| Conflicts | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

| | | | | |
|---|---|---|---|---|
| Dependencies | $\neg p_1 \lor p_2 \lor p_3$ | $\neg p_2 \lor p_3$ | $\neg p_3 \lor p_2$ | |
| Conflicts | $\neg p_4 \lor p_2$ | $\neg p_4 \lor p_3$ | $\neg p_1 \lor \neg p_4$ | $\neg p_3 \lor \neg p_4$ |
| Packages | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

| | | | | |
|---|---|---|---|---|
| Dependencies | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| Conflicts | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| Packages | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- $\varphi = (\neg p_1 \vee p_2 \vee p_3) \wedge (\neg p_2 \vee p_3) \wedge (\neg p_3 \vee p_2) \wedge (\neg p_4 \vee p_2) \wedge (\neg p_4 \vee p_3) \wedge (\neg p_1 \vee \neg p_4) \wedge (\neg p_3 \vee \neg p_4) \wedge (p_1) \wedge (p_2) \wedge (p_3) \wedge (p_4)$

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

| | | | | |
|---|---|---|---|---|
| Dependencies | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| Conflicts | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| Packages | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- Formula is unsatisfiable 
- We cannot install all packages
- How many packages can we install?

# Software Package Upgradeability Problem as SAT

Formula $\varphi$:

| | | | | |
|---|---|---|---|---|
| Dependencies | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| Conflicts | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| Packages | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- Formula is unsatisfiable 
- We cannot install all packages
- How many packages can we install?
- For example, we can install two packages. Can we do better?

# How to encode Software Package Upgradeability?

Software Package Upgradeability problem as MaxSAT:

- What are the hard constraints?
  -
- What are the soft constraints?
  -

# How to encode Software Package Upgradeability?

Software Package Upgradeability problem as MaxSAT:

- What are the hard constraints?
  - Dependencies and conflicts
- What are the soft constraints?
  - Installation of packages

# Software Package Upgradeability Problem as MaxSAT

MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| $S$ (Soft): | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- Dependencies and conflicts are encoded as hard clauses
- Installation of packages are encoded as soft clauses
- **Goal:** maximize the number of installed packages

# Software Package Upgradeability Problem as MaxSAT

MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| $S$ (Soft): | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- Dependencies and conflicts are encoded as hard clauses
- Installation of packages are encoded as soft clauses
- **Optimal solution** (3 out 4 packages are installed)

# Software Package Upgradeability Problem as MaxSAT

MaxSAT Formula:

| | | | | |
|---|---|---|---|---|
| $H$ (Hard): | $\neg p_1 \vee p_2 \vee p_3$ | $\neg p_2 \vee p_3$ | $\neg p_3 \vee p_2$ | |
| | $\neg p_4 \vee p_2$ | $\neg p_4 \vee p_3$ | $\neg p_1 \vee \neg p_4$ | $\neg p_3 \vee \neg p_4$ |
| $S$ (Soft): | $p_1$ | $p_2$ | $p_3$ | $p_4$ |

- Dependencies and conflicts are encoded as hard clauses
- Installation of packages are encoded as soft clauses
- **Optimal solution** (3 out 4 packages are installed)

Real-world applications use MaxSAT for this problem:

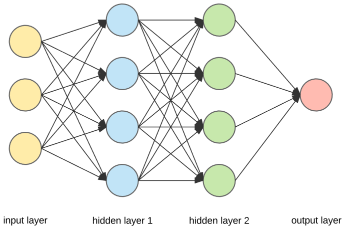
- Dependency management in Eclipse

[Berre and Rapicault, 2018]

# Real-World Applications of MaxSAT in Machine Learning

# Explainable Machine Learning

## Black Box (Classical) Model

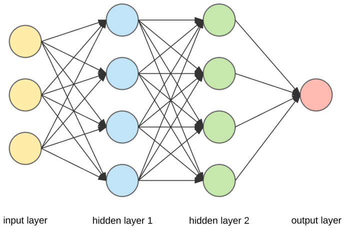



### Desirable Properties of ML models

- Accuracy (the decisions should be correct)

# Explainable Machine Learning

## Black Box (Classical) Model

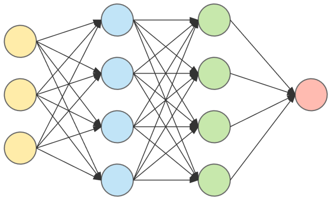



### Desirable Properties of ML models

- Accuracy (the decisions should be correct)
- Interpretability (users should be able to understand the reasoning)

# Explainable Machine Learning

Black Box (Classical) Model




Explainable Model

A sample is Iris Versicolor if
(sepal length $> 6.3$ **OR** sepal width $> 3$
**OR** petal width $\leq 1.5$)
**AND**
(sepal width $\leq 3$ **OR** petal length $> 4$
**OR** petal width $> 1.5$)
**AND**
(petal length $\leq 5$)

## Desirable Properties of ML models

- Accuracy (the decisions should be correct)
- Interpretability (users should be able to understand the reasoning)

# Using SAT and MaxSAT toward explainable ML

Two main research directions:

- **Explaining black box models** Ignatiev et al. [2019]; Narodytska et al. [2019, 2018]
- **Learning explainable models** Ignatiev et al. [2018b]; Maliotov and Meel [2018]; Ghosh and Meel [2019]

# Using SAT and MaxSAT toward explainable ML

Two main research directions:

- Explaining black box models Ignatiev et al. [2019]; Narodytska et al. [2019, 2018]
- Learning explainable models Ignatiev et al. [2018b]; Maliotov and Meel [2018]; Ghosh and Meel [2019]

## Case Study - Learning Interpretable Classification Rules with MaxSAT Maliotov and Meel [2018]; Ghosh and Meel [2019]

A MaxSAT based approach for learning classifiers that:

# Using SAT and MaxSAT toward explainable ML

Two main research directions:

- Explaining black box models Ignatiev et al. [2019]; Narodytska et al. [2019, 2018]
- Learning explainable models Ignatiev et al. [2018b]; Maliotov and Meel [2018]; Ghosh and Meel [2019]

## Case Study - Learning Interpretable Classification Rules with MaxSAT Maliotov and Meel [2018]; Ghosh and Meel [2019]

A MaxSAT based approach for learning classifiers that:

- Scales to datasets with thousands of points.
  - Especially in its incremental formulation

# Using SAT and MaxSAT toward explainable ML

Two main research directions:

- Explaining black box models Ignatiev et al. [2019]; Narodytska et al. [2019, 2018]
- Learning explainable models Ignatiev et al. [2018b]; Maliotov and Meel [2018]; Ghosh and Meel [2019]

## Case Study - Learning Interpretable Classification Rules with MaxSAT Maliotov and Meel [2018]; Ghosh and Meel [2019]

A MaxSAT based approach for learning classifiers that:

- Scales to datasets with thousands of points.
  - Especially in its incremental formulation
- Achieves accuracy comparable to other state-of-the-art methods (without sacrificing interpretability).

# Using SAT and MaxSAT toward explainable ML

Two main research directions:

- Explaining black box models Ignatiev et al. [2019]; Narodytska et al. [2019, 2018]
- Learning explainable models Ignatiev et al. [2018b]; Maliotov and Meel [2018]; Ghosh and Meel [2019]

## Case Study - Learning Interpretable Classification Rules with MaxSAT Maliotov and Meel [2018]; Ghosh and Meel [2019]

A MaxSAT based approach for learning classifiers that:

- Scales to datasets with thousands of points.
  - Especially in its incremental formulation
- Achieves accuracy comparable to other state-of-the-art methods (without sacrificing interpretability).
- We'd like to thank the authors for providing material!

# The problem

A sample is Iris Versicolor if:

(sepal length $> 6.3$ **OR** sepal width $> 3$ **OR** petal width $\leq 1.5$)

**AND**

(sepal width $\leq 3$ **OR** petal length $> 4$ **OR** petal width $> 1.5$)

# The problem

A sample is Iris Versicolor if:

(sepal length $> 6.3$ **OR** sepal width $> 3$ **OR** petal width $\leq 1.5$)

**AND**

(sepal width $\leq 3$ **OR** petal length $> 4$ **OR** petal width $> 1.5$)

| data | sepal length | sepal width | petal length | petal width |
|---|---|---|---|---|
| $D_1$ | 5.5 | 3.1 | 4.5 | 1.6 |
| $D_2$ | 3.4 | 3.1 | 3 | 1.1 |

# The problem

**Let:**
$x_1 = (\text{sepal length} > 6.3)$, $x_2 = (\text{sepal width} > 3)$,
$x_3 = (\text{petal length} > 4)$, $x_4 = (\text{petal width} > 1.5)$

# The problem

**Let:**
$x_1 = (\text{sepal length} > 6.3)$, $x_2 = (\text{sepal width} > 3)$,
$x_3 = (\text{petal length} > 4)$, $x_4 = (\text{petal width} > 1.5)$

A sample is Iris Versicolor if:
$(x_1 \textbf{ OR } x_2 \textbf{ OR } \neg x_4) \textbf{ AND } (\neg x_2 \textbf{ OR } x_3 \textbf{ OR } x_4)$

# The problem

**Let:**
$x_1 = (\text{sepal length} > 6.3)$, $x_2 = (\text{sepal width} > 3)$,
$x_3 = (\text{petal length} > 4)$, $x_4 = (\text{petal width} > 1.5)$

A sample is Iris Versicolor if:
$(x_1 \textbf{ OR } x_2 \textbf{ OR } \neg x_4) \textbf{ AND } (\neg x_2 \textbf{ OR } x_3 \textbf{ OR } x_4)$

| data | $x_1$ | $x_2$ | $x_3$ | $x_4$ |
|---|---|---|---|---|
| $D_1$ | 0 | 1 | 1 | 1 |
| $D_2$ | 0 | 1 | 0 | 0 |

# The problem

**Let:**
$x_1 = (\text{sepal length} > 6.3)$, $x_2 = (\text{sepal width} > 3)$,
$x_3 = (\text{petal length} > 4)$, $x_4 = (\text{petal width} > 1.5)$

A sample is Iris Versicolor if:

**($x_1$ OR $x_2$ OR $\neg x_4$) AND ($\neg x_2$ OR $x_3$ OR $x_4$)**

**A CNF-classifier**

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$$\mathcal{R} = (x_2 \vee x_3) \wedge (\neg x_1)$$

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2 \vee x_3) \wedge (\neg x_1)$

$\tau^1 = \{x_1 = 1, x_2 = 0, x_3 = 1\}$

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$$\mathcal{R} = (x_2 \lor x_3) \land (\neg x_1)$$

$\tau^1 = \{x_1 = 1, x_2 = 0, x_3 = 1\}$ $\quad \tau^{\mathbf{1}}(\mathcal{R}) = \mathbf{0}$

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2 \vee x_3) \wedge (\neg x_1)$

$\tau^1 = \{x_1 = 1, x_2 = 0, x_3 = 1\}$ $\tau^{\mathbf{1}}(\mathcal{R}) = \mathbf{0}$
$\tau^2 = \{x_1 = 0, x_2 = 1, x_3 = 1\}$

# Abstract Example

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2 \vee x_3) \wedge (\neg x_1)$

$\tau^1 = \{x_1 = 1, x_2 = 0, x_3 = 1\}$ $\boldsymbol{\tau^1}(\mathcal{R}) = \mathbf{0}$
$\tau^2 = \{x_1 = 0, x_2 = 1, x_3 = 1\}$ $\boldsymbol{\tau^2}(\mathcal{R}) = \mathbf{1}$

# Why Optimization?

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$$\mathcal{R} = (x_2 \vee x_3) \wedge (\neg x_1)$$

Classifiers are not unique.

# Why Optimization?

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2)$

Classifiers are not unique.

- Desirable properties:

# Why Optimization?

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2)$

$$\min \sum_{C \in \mathcal{R}} |C|$$

Classifiers are not unique.

- Desirable properties:
  - Explainability

# Why Optimization?

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$\mathcal{R} = (x_2)$

$$\min \sum_{C \in \mathcal{R}} |C| + \lambda \sum_i \epsilon_i$$

where $\epsilon_i = 1$ iff $d_i$ is considered noise
i.e. $\tau^i(\mathcal{R}) \neq y_i$

## Classifiers are not unique.

- Desirable properties:
  - Explainability
  - Accuracy

# Explainable Classifiers with MaxSAT

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
$\#$clauses $k$

# Explainable Classifiers with MaxSAT

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
#clauses $k$

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$cost(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

# Explainable Classifiers with MaxSAT

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$\mathit{cost}(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
#clauses $k$

**Main Variables (of $F$):**

# Explainable Classifiers with MaxSAT

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
$\#$clauses $k$

**Main Variables (of $F$):**
$\mathbf{b}_\mathbf{i}^\mathbf{t}$, $t = 1 \ldots m$, $i = 1 \ldots k$

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$cost(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

# Explainable Classifiers with MaxSAT

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$cost(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
#clauses $k$

**Main Variables (of $F$):**
$\mathbf{b_i^t}$, $t = 1 \ldots m$, $i = 1 \ldots k$

$\tau(b_i^t) = 1$ if $x_t \in C_i \in \mathcal{R}$

# Explainable Classifiers with MaxSAT

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
$\#$clauses $k$

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$cost(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

**Main Variables (of $F$):**
$\mathbf{b_i^t}$, $t = 1 \ldots m$, $i = 1 \ldots k$
$\eta_\mathbf{i}$, $i = 1 \ldots n$

$\tau(b_i^t) = 1$ if $x_t \in C_i \in \mathcal{R}$

# Explainable Classifiers with MaxSAT

*Input:*
Data:
$\mathcal{D} = \{(\mathbf{d}_1, y_1), \ldots, (\mathbf{d}_m, y_m)\}$
noise weight $\lambda$,
$\#$clauses $k$

*Goal:*
MaxSAT Instance $F$
s.t. each solution $\tau$
corresponds to
a CNF classifier
$\mathcal{R}^\tau = \bigwedge_{i=1}^{k} C_i$ with
$cost(\tau) = \sum_{C \in \mathcal{R}^\tau} |C| + \lambda \sum_i \epsilon_i$

**Main Variables (of $F$):**

$\mathbf{b_i^t}$, $t = 1 \ldots m$, $i = 1 \ldots k$ — $\tau(b_i^t) = 1$ if $x_t \in C_i \in \mathcal{R}$

$\eta_{\mathbf{i}}$, $i = 1 \ldots n$ — $\tau(\eta_i) = 1$ if $\mathbf{d}_i$ is noise

# Clauses in $F$

### Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \rightarrow \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i \text{ satisfies } C_j)$.

If $y_i = 0$ include $\neg\eta_i \rightarrow \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i \text{ falsifies } C_j)$.

Where $\tau^i = \{x_1 = d_1, \ldots x_m = d_m\}$

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \to \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i$ satisfies $C_j)$.

If $y_i = 0$ include $\neg\eta_i \to \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i$ falsifies $C_j)$.

## Soft Clauses:

Capture the cost function: $\min \sum_{C \in \mathcal{R}} |C| + \lambda \sum_i \epsilon_i$

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \rightarrow \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i$ satisfies $C_j)$.

If $y_i = 0$ include $\neg\eta_i \rightarrow \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i$ falsifies $C_j)$.

## Soft Clauses:

Capture the cost function: $\min \sum_{C \in \mathcal{R}} |C| + \lambda \sum_i \epsilon_i$

Considering $(\mathbf{d}_i, y_i)$ as noise incurs a cost of $\lambda$:

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \rightarrow \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i$ satisfies $C_j)$.

If $y_i = 0$ include $\neg\eta_i \rightarrow \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i$ falsifies $C_j)$.

## Soft Clauses:

Capture the cost function: $\min \sum_{C\in\mathcal{R}} |C| + \lambda \sum_i \epsilon_i$

Considering $(\mathbf{d}_i, y_i)$ as noise incurs a cost of $\lambda$:

$(\neg\eta_i)$, with weight $\lambda$

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \rightarrow \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i \text{ satisfies } C_j)$.

If $y_i = 0$ include $\neg\eta_i \rightarrow \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i \text{ falsifies } C_j)$.

## Soft Clauses:

Capture the cost function: min $\sum_{C \in \mathcal{R}} |C| + \lambda \sum_i \epsilon_i$

Considering $(\mathbf{d}_i, y_i)$ as noise incurs a cost of $\lambda$:

$(\neg\eta_i)$, with weight $\lambda$

Adding any literal $x_t$ to $C_i$ incurs a cost of 1:

# Clauses in $F$

## Hard Clauses:

Any data point $(\mathbf{d}_i, y_i)$ is either noise or correctly classified:

If $y_i = 1$ include $\neg\eta_i \rightarrow \bigwedge_{j=1}^{k} \mathrm{CNF}(\tau^i$ satisfies $C_j)$.

If $y_i = 0$ include $\neg\eta_i \rightarrow \bigvee_{j=1}^{k} \mathrm{CNF}(\tau^i$ falsifies $C_j)$.

## Soft Clauses:

Capture the cost function: $\min \sum_{C \in \mathcal{R}} |C| + \lambda \sum_i \epsilon_i$

Considering $(\mathbf{d}_i, y_i)$ as noise incurs a cost of $\lambda$:

$(\neg\eta_i)$, with weight $\lambda$

Adding any literal $x_t$ to $C_i$ incurs a cost of 1:

$(\neg b_i^t)$ with weight 1

# Example

## Data:

$d_1 = (1, 0, 1)$, $y_1 = 0$
$d_2 = (0, 1, 1)$, $y_2 = 1$

$k = 2$, $\lambda = 2$

# Example

Data:

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$k = 2,\ \lambda = 2$

Hard Clauses

- $\neg\eta_1 \rightarrow \neg(b_1^1 \vee b_1^3) \vee \neg(b_2^1 \vee b_2^3)$
- $\neg\eta_2 \rightarrow (b_1^2 \vee b_1^3) \wedge (b_2^2 \vee b_2^3)$

# Example

## Data:

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$k = 2,\ \lambda = 2$

## Hard Clauses

- $\neg\eta_1 \to \neg(b_1^1 \vee b_1^3) \vee \neg(b_2^1 \vee b_2^3)$
- $\neg\eta_2 \to (b_1^2 \vee b_1^3) \wedge (b_2^2 \vee b_2^3)$

## Soft clauses

- $(\neg b_1^1), (\neg b_1^2), (\neg b_1^3), (\neg b_2^1), (\neg b_2^2), (\neg b_2^3)$ weight 1
- $(\neg\eta_1), (\neg\eta_2)$ weight 2.

# Example

## Data:

$d_1 = (1, 0, 1),\ y_1 = 0$
$d_2 = (0, 1, 1),\ y_2 = 1$

$$\mathcal{R}^\tau = (x_2) \wedge (x_2)$$

$k = 2,\ \lambda = 2$

## Hard Clauses

- $\neg\eta_1 \rightarrow \neg(b_1^1 \vee b_1^3) \vee \neg(b_2^1 \vee b_2^3)$
- $\neg\eta_2 \rightarrow (b_1^2 \vee b_1^3) \wedge (b_2^2 \vee b_2^3)$

## Soft clauses

- $(\neg b_1^1), (\neg b_1^2), (\neg b_1^3), (\neg b_2^1), (\neg b_2^2), (\neg b_2^3)$ weight 1
- $(\neg\eta_1), (\neg\eta_2)$ weight 2.

# More on Modelling with MaxSAT

# Representing High-level Soft Constraints

## Basic Idea

Finite-domain soft constraint $\mathcal{C}$ with associated weight $W_{\mathcal{C}}$.

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

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

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

Important for various applications of MaxSAT

# Handling Non-Integer Weights

## Problem

MaxSAT supports (by definition and input format) only integer weights on soft clauses. *The objective function of my problem has real-valued weights.*

## Solution

Scale weight range to 64-bit representation range & truncate to integers.

- Standard trick when applying MaxSAT solvers
- Solvers less and less volative in terms of large weights
- *While some accuracy may be lost, similar issues are standardly seen e.g. when applying mixed-linear integer programming*

# Tools for Modelling and Building Solvers

## Preprocessing: simplifying encodings before solving

- MaxPre [Korhonen, Berg, Saikko, and Järvisalo, 2017]
- Coprocessor [Manthey, 2012]

## Automated encoding of high-level constraints

- MaxPre: extends input language to cardinality constraints
- *Room for improvement in terms of easy-to-use tools!*

## PySAT: Python library for prototyping solvers

- Offers easy interfacing with SAT solvers [Ignatiev, Morgado, and Marques-Silva, 2018a]
- Cardinality constraint support built-in
- Efficient: one of the most recent efficient core-guided solvers, RC2, is PySAT-based

# Applying MaxSAT to New Domains

- *How to model problem X as* MaxSAT*?*
  - Developing compact encodings
  - Redundant constraints via insights into the problem domain
  - Representation of weights
  - ...
- *Understanding the interplay between* encodings *and* solver techniques
  - Encodings: compactness vs. propagation
  - Underlying core-structure of encodings
  - The "best" solvers for current benchmark sets *may not be best* for novel applications!
    - Requires trial-and-error & in-depth understanding of solvers and the problem domain

# Summary

# MaxSAT

- Low-level constraint language:
  weighted Boolean combinations of binary variables
  - Gives tight control over how exactly to encode problem
- Exact optimization: provably optimal solutions
- MaxSAT solvers:
  - build on top of highly efficient SAT solver technology
  - various alternative approaches:
    branch-and-bound, model-improving, core-guided, IHS, ...
  - standard WCNF input format
  - yearly MaxSAT solver evaluations

## Success of MaxSAT

- Attractive alternative to other constrained optimization paradigms
- Number of applications increasing
- Solver technology improving rapidly

# Topics Covered

- Basic concepts
- Survey of some of the currently most relevant solving algorithms
  - model-improving
  - core-guided
  - SAT-IP hybrids based on the implicit hitting set approach
  - incomplete solving
- Modelling with MaxSAT
  - ideas for how to encode different problems as MaxSAT
  - understanding some of the benefits of using MaxSAT

# Further Topics and Research Directions

## Incomplete Solving

Quick recent progress suggests that further improvements are to be expected

## Preprocessing

*How to simplify* MaxSAT *instances to make them easier for solver(s)?*

- Recent progress:
  - Lifting SAT-based techniques [Belov, Morgado, and Marques-Silva, 2013; Berg and Järvisalo, 2019]
  - Native MaxSAT techniques [Berg, Saikko, and Järvisalo, 2015b,a, 2016; Korhonen, Berg, Saikko, and Järvisalo, 2017]
  - Analysis [Berg and Järvisalo, 2016, 2019]
- Challenge: effective integration with MaxSAT algorithms
  - Inprocessing MaxSAT solving?
    (In analogy to SAT [Järvisalo, Heule, and Biere, 2012])

# Further Topics and Research Directions

## Parallel Solving

*How to truly make use of massively parallel computing infrastructures for MaxSAT?*

- Obtaining linear speed-ups (or even more) turned out to be highly non-trivial to obtain, similarly as in SAT solving
- Some progress, but much more unleashed potential
  [Martins, Manquinho, and Lynce, 2011, 2012; van der Tak, Heule, and Biere, 2012; Terra-Neves, Lynce, and Manquinho, 2016]

## Support for Incremental Computations

*Solving several related instances without computing from scratch*

- Solving huge MaxSAT instances
- Applying MaxSAT to solve beyond-NP optimization problems
- Applications benefiting from incremental computations
- Currently, few solvers offer (restricted) incremental APIs

[Saikko, Berg, and Järvisalo, 2016]

# Further Reading and Links

## Surveys

- "Maximum Satisfiability" by Bacchus, Järvisalo & Martins
  - Chapter in forthcoming vol. 2 of Handbook of Satisfiability
  - Preprint available, link on tutorial webpage
- Somewhat older surveys:
  - Handbook chapter on MaxSAT: [Li and Manyà, 2009]
  - Surveys on MaxSAT algorithms:

    [Ansótegui, Bonet, and Levy, 2013a]

    [Morgado, Heras, Liffiton, Planes, and Marques-Silva, 2013a]

## MaxSAT Evaluations

https://maxsat-evaluations.github.io

Most recent report: [Bacchus, Järvisalo, and Martins, 2019]

Thank you for attending!

# Bibliography I


Roberto Javier Asín Achá and Robert Nieuwenhuis. Curriculum-based course timetabling with SAT and MaxSAT. *Annals of Operations Research*, 218(1):71–91, 2014. doi: 10.1007/s10479-012-1081-x. URL https://doi.org/10.1007/s10479-012-1081-x.

Mario Alviano, Carmine Dodaro, and Francesco Ricca. A MaxSAT algorithm using cardinality constraints of bounded size. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2677–2683. AAAI Press, 2015. URL http://ijcai.org/papers15/Abstracts/IJCAI15-379.html.

Benjamin Andres, Benjamin Kaufmann, Oliver Matheis, and Torsten Schaub. Unsatisfiability-based optimization in clasp. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012, September 4-8, 2012, Budapest, Hungary*, volume 17 of *LIPIcs*, pages 211–221. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSAT through satisfiability testing. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 427–440. Springer, 2009.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. A new algorithm for weighted partial MaxSAT. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

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

Carlos Ansótegui, Idelfonso Izquierdo, Felip Manyà, and José Torres-Jiménez. A Max-SAT-based approach to constructing optimal covering arrays. In Karina Gibert, Vicent J. Botti, and Ramón Reig Bolaño, editors, *Artificial Intelligence Research and Development - Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence, Vic, Catalonia, Spain, October 23-25, 2013.*, volume 256 of *Frontiers in Artificial Intelligence and Applications*, pages 51–59. IOS Press, 2013b.

Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in MaxSAT. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 283–289. AAAI Press, 2015. ISBN 978-1-57735-738-4.

# Bibliography II


Josep Argelich, Inês Lynce, and João Marques-Silva. On solving boolean multilevel optimization problems. In Craig Boutilier, editor, *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 393–398. AAAI Press, 2009.

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

Florent Avellaneda. UWrMaxSat - a new MiniSat+-based Solver in MaxSAT Evaluation 2020. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, volume B-2020-2 of *Department of Computer Science Series of Publications B*, pages 8–9. University of Helsinki, 2020.

Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017. doi: 10.1007/978-3-319-66158-2_41. URL https://doi.org/10.1007/978-3-319-66158-2_41.

Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maxsat evaluation 2018: New developments and detailed results. *J. Satisf. Boolean Model. Comput.*, 11(1):99–131, 2019. doi: 10.3233/SAT190119. URL https://doi.org/10.3233/SAT190119.

Amine Belabed, Esma Aïmeur, Mohammed Amine Chikh, and Hadjila Fethallah. A privacy-preserving approach for composite web service selection. *Transactions on Data Privacy*, 10(2):83–115, 2017. URL http://www.tdp.cat/issues16/abs.a253a16.php.

Anton Belov, António Morgado, and João Marques-Silva. SAT-based preprocessing for MaxSAT. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 96–111. Springer, 2013.

Jeremias Berg and Matti Järvisalo. Optimal correlation clustering via MaxSAT. In Wei Ding, Takashi Washio, Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu, editors, *13th IEEE International Conference on Data Mining Workshops, ICDM Workshops, TX, USA, December 7-10, 2013*, pages 750–757. IEEE Computer Society, 2013.

# Bibliography III


Jeremias Berg and Matti Järvisalo. SAT-based approaches to treewidth computation: An evaluation. In *26th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2014, Limassol, Cyprus, November 10-12, 2014*, pages 328–335. IEEE Computer Society, 2014.

Jeremias Berg and Matti Järvisalo. Impact of SAT-based preprocessing on core-guided MaxSAT solving. In Michel Rueher, editor, *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, volume 9892 of *Lecture Notes in Computer Science*, pages 66–85. Springer, 2016. doi: 10.1007/978-3-319-44953-1_5. URL https://doi.org/10.1007/978-3-319-44953-1_5.

Jeremias Berg and Matti Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 244:110–142, 2017. doi: 10.1016/j.artint.2015.07.001. URL https://doi.org/10.1016/j.artint.2015.07.001.

Jeremias Berg and Matti Järvisalo. Unifying reasoning and core-guided search for maximum satisfiability. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2019. doi: 10.1007/978-3-030-19570-0_19. URL https://doi.org/10.1007/978-3-030-19570-0_19.

Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth Bayesian networks via maximum satisfiability. In Jukka Corander and Samuel Kaski, editors, *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 86–95. JMLR.org, 2014.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Re-using auxiliary variables for MaxSAT preprocessing. In *27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015*, pages 813–820. IEEE Computer Society, 2015a.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Improving the effectiveness of SAT-based preprocessing for MaxSAT. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 239–245. AAAI Press, 2015b. URL http://ijcai.org/papers15/Abstracts/IJCAI15-040.html.

# Bibliography IV


Jeremias Berg, Paul Saikko, and Matti Järvisalo. Subsumed label elimination for maximum satisfiability. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 630–638. IOS Press, 2016. doi: 10.3233/978-1-61499-672-9-630. URL https://doi.org/10.3233/978-1-61499-672-9-630.

Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. Core-boosted linear search for incomplete maxsat. In Louis-Martin Rousseau and Kostas Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*, volume 11494 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2019. doi: 10.1007/978-3-030-19212-9_3. URL https://doi.org/10.1007/978-3-030-19212-9_3.

Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set maxsat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi: 10.1007/978-3-030-51825-7_20. URL https://doi.org/10.1007/978-3-030-51825-7_20.

Daniel Le Berre and Pascal Rapicault. Boolean-based dependency management for the eclipse ecosystem. *Int. J. Artif. Intell. Tools*, 27(1):1840003:1–1840003:23, 2018. doi: 10.1142/S0218213018400031. URL https://doi.org/10.1142/S0218213018400031.

Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. MaxSAT-based scheduling of B2B meetings. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2015.

Kerstin Bunte, Matti Järvisalo, Jeremias Berg, Petri Myllymäki, Jaakko Peltonen, and Samuel Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 1694–1700. AAAI Press, 2014. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8242.

# Bibliography V


Shaowei Cai, Chuan Luo, Jinkun Lin, and Kaile Su. New local search methods for partial MaxSAT. *Artificial Intelligence*, 240:1–18, 2016. doi: 10.1016/j.artint.2016.07.006. URL https://doi.org/10.1016/j.artint.2016.07.006.

Yibin Chen, Sean Safarpour, Andreas G. Veneris, and João Marques-Silva. Spatial and temporal design debug using partial MaxSAT. In Fabrizio Lombardi, Sanjukta Bhanja, Yehia Massoud, and R. Iris Bahar, editors, *Proceedings of the 19th ACM Great Lakes Symposium on VLSI 2009, Boston Area, MA, USA, May 10-12 2009*, pages 345–350. ACM, 2009.

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

Eldan Cohen, Guoyu Huang, and J. Christopher Beck. (I can get) satisfaction: Preference-based scheduling for concert-goers at multi-venue music festivals. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2017.

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

Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013a.

Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSAT. In Matti Järvisalo and Allen Van Gelder, editors, *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2013b.

# Bibliography VI


Ricardo Tavares de Oliveira and Fabiano Silva. On a relative MaxSAT encoding for the steiner tree problem in graphs. In Obdulia Pichardo-Lagunas, Oscar Herrera-Alcántara, and Gustavo Arroyo-Figueroa, editors, *Advances in Artificial Intelligence and Its Applications - 14th Mexican International Conference on Artificial Intelligence, MICAI 2015, Cuernavaca, Morelos, Mexico, October 25-31, 2015. Proceedings, Part II*, volume 9414 of *Lecture Notes in Computer Science*, pages 422–434. Springer, 2015.

Emir Demirovic and Nysret Musliu. MaxSAT-based large neighborhood search for high school timetabling. *Computers & Operations Research*, 78:172–180, 2017. doi: 10.1016/j.cor.2016.08.004. URL `https://doi.org/10.1016/j.cor.2016.08.004`.

Emir Demirovic and Peter J. Stuckey. Techniques inspired by local search for incomplete maxsat and the linear algorithm: Varying resolution and solution-guided search. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2019. doi: 10.1007/978-3-030-30048-7_11. URL `https://doi.org/10.1007/978-3-030-30048-7_11`.

Emir Demirović, Nysret Musliu, and Felix Winter. Modeling and solving staff scheduling with partial weighted MaxSAT. *Annals of Operations Research*, 2017.

Rayna Dimitrova, Mahsa Ghasemi, and Ufuk Topcu. Maximum realizability for linear temporal logic specifications. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 458–475. Springer, 2018. doi: 10.1007/978-3-030-01090-4_27. URL `https://doi.org/10.1007/978-3-030-01090-4_27`.

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

# Bibliography VII


Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. Automated synthesis of semantic malware signatures using maximum satisfiability. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. URL https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/automated-synthesis-semantic-malware-signatures-using-maximum-satisfiability/.

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

Bishwamittra Ghosh and Kuldeep S. Meel. IMLI: an incremental framework for maxsat-based learning of interpretable classification rules. In Vincent Conitzer, Gillian K. Hadfield, and Shannon Vallor, editors, *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society, AIES 2019, Honolulu, HI, USA, January 27-28, 2019*, pages 203–210. ACM, 2019. doi: 10.1145/3306618.3314283. URL https://doi.org/10.1145/3306618.3314283.

Ana Graça, João Marques-Silva, and Inês Lynce. Haplotype inference using propositional satisfiability. In Renato Bruni, editor, *Mathematical Approaches to Polymer Sequence Analysis and Related Problems*, pages 127–147. Springer, 2011a.

Ana Graça, João Marques-Silva, Inês Lynce, and Arlindo L. Oliveira. Haplotype inference with pseudo-boolean optimization. *Annals of Operations Research*, 184(1):137–162, 2011b.

João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In Michela Milano, editor, *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*, volume 7514 of *Lecture Notes in Computer Science*, pages 941–956. Springer, 2012.

Federico Heras, António Morgado, and João Marques-Silva. Core-guided binary search algorithms for maximum satisfiability. In Wolfram Burgard and Dan Roth, editors, *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. AAAI Press, 2011.

Wenxuan Huang, Daniil A. Kitchaev, Stephen T. Dacek, Ziqin Rong, Alexander Urban, Shan Cao, Chuan Luo, and Gerbrand Ceder. Finding and proving the exact ground state of a generalized ising model by convex optimization and MAX-SAT. *Physical Review B*, 94:134424, 2016. doi: 10.1103/PhysRevB.94.134424. URL https://link.aps.org/doi/10.1103/PhysRevB.94.134424.

# Bibliography VIII


Antti Hyttinen, Sergey M. Plis, Matti Järvisalo, Frederick Eberhardt, and David Danks. A constraint optimization approach to causal discovery from subsampled time series data. *International Journal of Approximate Reasoning*, 90:208–225, 2017a. doi: 10.1016/j.ijar.2017.07.009. URL `https://doi.org/10.1016/j.ijar.2017.07.009`.

Antti Hyttinen, Paul Saikko, and Matti Järvisalo. A core-guided approach to learning optimal causal graphs. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 645–651. ijcai.org, 2017b. doi: 10.24963/ijcai.2017/90. URL `https://doi.org/10.24963/ijcai.2017/90`.

Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Towards efficient optimization in package management systems. In Pankaj Jalote, Lionel C. Briand, and André van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 745–755. ACM, 2014.

Alexey Ignatiev, António Morgado, and João Marques-Silva. PySAT: A python toolkit for prototyping with SAT oracles. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018a. doi: 10.1007/978-3-319-94144-8_26. URL `https://doi.org/10.1007/978-3-319-94144-8_26`.

Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva. A SAT-based approach to learn explainable decision sets. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018b. doi: 10.1007/978-3-319-94205-6_41. URL `https://doi.org/10.1007/978-3-319-94205-6_41`.

Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1511–1519. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33011511. URL `https://doi.org/10.1609/aaai.v33i01.33011511`.

# Bibliography IX


Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012. doi: 10.1007/978-3-642-31365-3_28. URL https://doi.org/10.1007/978-3-642-31365-3_28.

M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In Mary W. Hall and David A. Padua, editors, *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 437–446. ACM, 2011.

Saurabh Joshi, Prateek Kumar, Ruben Martins, and Sukrut Rao. Approximation strategies for incomplete MaxSAT. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 219–228. Springer, 2018. doi: 10.1007/978-3-319-98334-9_15. URL https://doi.org/10.1007/978-3-319-98334-9_15.

Tuukka Korhonen and Matti Järvisalo. Finding most compatible phylogenetic trees over multi-state characters. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1544–1551. AAAI Press, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5514.

Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo. MaxPre: An extended MaxSAT preprocessor. In Serge Gaspers and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing - SAT 2017 - 20th International Conference, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10491 of *Lecture Notes in Computer Science*, pages 449–456. Springer, 2017. doi: 10.1007/978-3-319-66263-3_28. URL https://doi.org/10.1007/978-3-319-66263-3_28.

Miyuki Koshimura, Tong Zhang, Hiroshi Fujita, and Ryuzo Hasegawa. QMaxSAT: A partial Max-SAT solver. *Journal of Satisfiability, Boolean Modeling and Computation*, 8(1/2):95–100, 2012.

Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability*, pages 613–631. 2009. doi: 10.3233/978-1-58603-929-5-613. URL http://dx.doi.org/10.3233/978-1-58603-929-5-613.

Chu Min Li and Zhe Quan. An efficient branch-and-bound algorithm based on MaxSAT for the maximum clique problem. In Maria Fox and David Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

# Bibliography X


Chu-Min Li, Hua Jiang, and Ruchu Xu. Incremental MaxSAT reasoning to reduce branches in a branch-and-bound algorithm for MaxClique. In Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion, editors, *Learning and Intelligent Optimization - 9th International Conference, LION 9, Lille, France, January 12-15, 2015. Revised Selected Papers*, volume 8994 of *Lecture Notes in Computer Science*, pages 268–274. Springer, 2015.

Xiaojuan Liao, Hui Zhang, and Miyuki Koshimura. Reconstructing AES key schedule images with SAT and MaxSAT. *IEICE Transactions*, 99-D(1):141–150, 2016. doi: 10.1587/transinf.2015EDP7223. URL `https://doi.org/10.1587/transinf.2015EDP7223`.

Pey-Chang Kent Lin and Sunil P Khatri. Application of Max-SAT-based ATPG to optimal cancer therapy design. *BMC Genomics*, 13, 2012.

Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: an efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017. doi: 10.1016/j.artint.2016.11.001. URL `https://doi.org/10.1016/j.artint.2016.11.001`.

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

Dmitry Maliotov and Kuldeep S. Meel. MLIC: A MaxSAT-based framework for learning interpretable classification rules. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2018.

Ravi Mangal, Xin Zhang, Aditya V. Nori, and Mayur Naik. A user-guided approach to program analysis. In Elisabetta Di Nitto, Mark Harman, and Patrick Heymans, editors, *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 462–473. ACM, 2015.

Vasco M. Manquinho, João Marques-Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009. doi: 10.1007/978-3-642-02777-2_45. URL `http://dx.doi.org/10.1007/978-3-642-02777-2_45`.

# Bibliography XI


Norbert Manthey. Coprocessor 2.0 - A flexible CNF simplifier - (tool presentation). In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 436–441. Springer, 2012. doi: 10.1007/978-3-642-31612-8_34. URL https://doi.org/10.1007/978-3-642-31612-8_34.

Felip Manyà, Santiago Negrete, Carme Roig, and Joan Ramon Soler. A MaxSAT-based approach to the team composition problem in a classroom. In Gita Sukthankar and Juan A. Rodríguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Visionary Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*, volume 10643 of *Lecture Notes in Computer Science*, pages 164–173. Springer, 2017.

Tiepelt Marcel Kevin and Singh Tilak Raj. Finding pre-production vehicle configurations using a MaxSAT framework. In *Proceedings of the 18th International Configuration Workshop*, pages 117–122. École des Mines d'Albi-Carmaux, 2016.

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

João Marques-Silva, Mikolás Janota, Alexey Ignatiev, and António Morgado. Efficient model based diagnosis with maximum satisfiability. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1966–1972. AAAI Press, 2015.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Exploiting cardinality encodings in parallel maximum satisfiability. In *IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011, Boca Raton, FL, USA, November 7-9, 2011*, pages 313–320. IEEE Computer Society, 2011. ISBN 978-1-4577-2068-0. URL http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6101101.

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

Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSAT. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 531–548. Springer, 2014a.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 438–445. Springer, 2014b.

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

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

António Morgado, Mark H. Liffiton, and João Marques-Silva. MaxSAT-based MCS enumeration. In Armin Biere, Amir Nahir, and Tanja E. J. Vos, editors, *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 86–101. Springer, 2013b.

António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In Barry O'Sullivan, editor, *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7. URL http://dx.doi.org/10.1007/978-3-319-10428-7.

Christian J. Muise, J. Christopher Beck, and Sheila A. McIlraith. Optimal partial-order plan relaxation via MaxSAT. *Journal of Artificial Intelligence Research*, 57:113–149, 2016. doi: 10.1613/jair.5128. URL https://doi.org/10.1613/jair.5128.

Alexander Nadel. Solving MaxSAT with bit-vector optimization. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2018.

Alexander Nadel. Anytime weighted maxsat with improved polarity selection and bit-vector optimization. In Clark W. Barrett and Jin Yang, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 193–202. IEEE, 2019. doi: 10.23919/FMCAD.2019.8894273. URL https://doi.org/10.23919/FMCAD.2019.8894273.

# Bibliography XIII

Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSAT resolution. In Carla E. Brodley and Peter Stone, editors, *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, pages 2717–2723. AAAI Press, 2014. ISBN 978-1-57735-661-5. URL http://www.aaai.org/Library/AAAI/aaai14contents.php.

Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6615–6624. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16898.

Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and João Marques-Silva. Assessing heuristic machine learning explanations with model counting. In Mikolás Janota and Inês Lynce, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 267–278. Springer, 2019. doi: 10.1007/978-3-030-24258-9_19. URL https://doi.org/10.1007/978-3-030-24258-9_19.

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

Andreas Niskanen, Johannes Peter Wallner, and Matti Järvisalo. Synthesizing argumentation frameworks from examples. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 551–559. IOS Press, 2016a. doi: 10.3233/978-1-61499-672-9-551. URL https://doi.org/10.3233/978-1-61499-672-9-551.

# Bibliography XIV


Andreas Niskanen, Johannes Peter Wallner, and Matti Järvisalo. Optimal status enforcement in abstract argumentation. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1216–1222. IJCAI/AAAI Press, 2016b. URL `http://www.ijcai.org/Abstract/16/176`.

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

Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In Olaf Beyersdorff and Christoph M. Wintersteiger, editors, *Theory and Applications of Satisfiability Testing - SAT 2018 - 21st International Conference, SAT 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10929 of *Lecture Notes in Computer Science*, pages 37–53. Springer, 2018.

Marek Piotrow. UWrMaxSat - a new MiniSat+-based Solver in MaxSAT Evaluation 2019. In *MaxSAT Evaluation 2019: Solver and Benchmark Descriptions*, volume B-2019-2 of *Department of Computer Science Series of Publications B*, page 11. University of Helsinki, 2019.

Sean Safarpour, Hratch Mangassarian, Andreas G. Veneris, Mark H. Liffiton, and Karem A. Sakallah. Improved design debugging using maximum satisfiability. In *Formal Methods in Computer-Aided Design, 7th International Conference, FMCAD 2007, Austin, Texas, USA, November 11-14, 2007, Proceedings*, pages 13–19. IEEE Computer Society, 2007. doi: 10.1109/FAMCAD.2007.26. URL `https://doi.org/10.1109/FAMCAD.2007.26`.

Paul Saikko, Brandon Malone, and Matti Järvisalo. MaxSAT-based cutting planes for learning graphical models. In Laurent Michel, editor, *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*, volume 9075 of *Lecture Notes in Computer Science*, pages 347–356. Springer, 2015.

Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 539–546. Springer, 2016.

# Bibliography XV


Ahmad Shabani and Bijan Alizadeh. PMTP: A MAX-SAT based approach to detect hardware trojan using propagation of maximum transition probability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018. doi: 10.1109/TCAD.2018.2889663.

Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik. Maximum satisfiability in software analysis: Applications and techniques. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10427 of *Lecture Notes in Computer Science*, pages 68–94. Springer, 2017.

Miguel Terra-Neves, Inês Lynce, and Vasco M. Manquinho. Non-portfolio approaches for distributed maximum satisfiability. In *28th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2016, San Jose, CA, USA, November 6-8, 2016*, pages 436–443. IEEE Computer Society, 2016. doi: 10.1109/ICTAI.2016.0073. URL `https://doi.org/10.1109/ICTAI.2016.0073`.

Peter van der Tak, Marijn Heule, and Armin Biere. Concurrent cube-and-conquer - (poster presentation). In Alessandro Cimatti and Roberto Sebastiani, editors, *Theory and Applications of Satisfiability Testing - SAT 2012 - 15th International Conference, Trento, Italy, June 17-20, 2012. Proceedings*, volume 7317 of *Lecture Notes in Computer Science*, pages 475–476. Springer, 2012.

Abderrahim Ait Wakrime, Said Jabbour, and Nabil Hameurlain. A MaxSAT based approach for QoS cloud services. *International Journal of Parallel, Emergent and Distributed Systems*, 2018.

Johannes Peter Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. *Journal of Artificial Intelligence Research*, 60:1–40, 2017. doi: 10.1613/jair.5415. URL `https://doi.org/10.1613/jair.5415`.

Guneshi T. Wickramaarachchi, Wahbeh H. Qardaji, and Ninghui Li. An efficient framework for user authorization queries in RBAC systems. In Barbara Carminati and James Joshi, editors, *14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, Stresa, Italy, June 3-5, 2009, Proceedings*, pages 23–32. ACM, 2009. doi: 10.1145/1542207.1542213. URL `https://doi.org/10.1145/1542207.1542213`.

Hui Xu, R. A. Rutenbar, and K. Sakallah. sub-SAT: a formulation for relaxed Boolean satisfiability with applications in routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22 (6):814–820, 2003.

# Bibliography XVI


Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In Jörg Hoffmann and Bart Selman, editors, *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*. AAAI Press, 2012.

Xin Zhang, Ravi Mangal, Aditya V. Nori, and Mayur Naik. Query-guided maximum satisfiability. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 109–122. ACM, 2016. doi: 10.1145/2837614.2837658. URL https://doi.org/10.1145/2837614.2837658.

Charlie Shucheng Zhu, Georg Weissenbacher, and Sharad Malik. Post-silicon fault localisation using maximum satisfiability and backbones. In Per Bjesse and Anna Slobodová, editors, *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*, pages 63–66. FMCAD Inc., 2011.