Maximum Satisfiability Solving

Jeremias Berg Matti Järvisalo

University of Helsinki
Finland

April 13, 2021 Simons Institute / Online

Many thanks for Fahiem Bacchus and Ruben Martins for contribution to earlier versions of these slides!

Maximum Satisfiability

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

Outline

1. Motivation and basic concepts
2. MaxSAT solving:
   Practical algorithms for $\textsc{MaxSAT}$

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

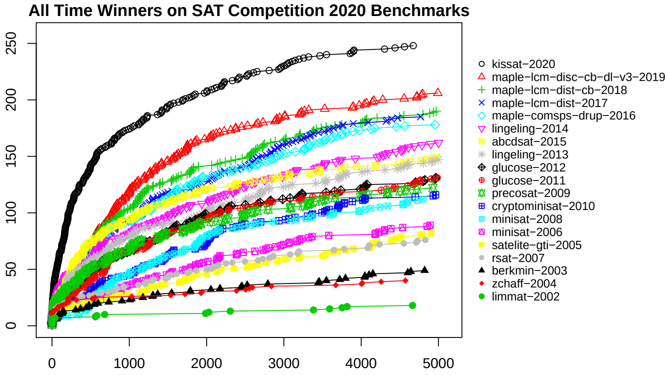
Not merely a central problem in *theory*:

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

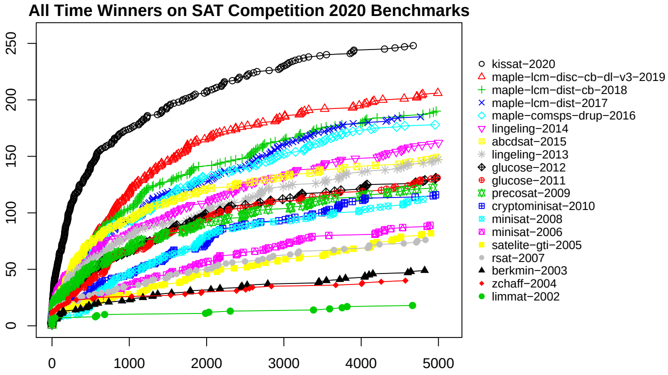
From 100s of variables and constraints (early 90s)
up to 10M variables and constraints. (21st century).



Plot provided by Armin Biere

SAT Solvers

From 100s of variables and constraints (early 90s)
up to 10M variables and constraints. (21st century).



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

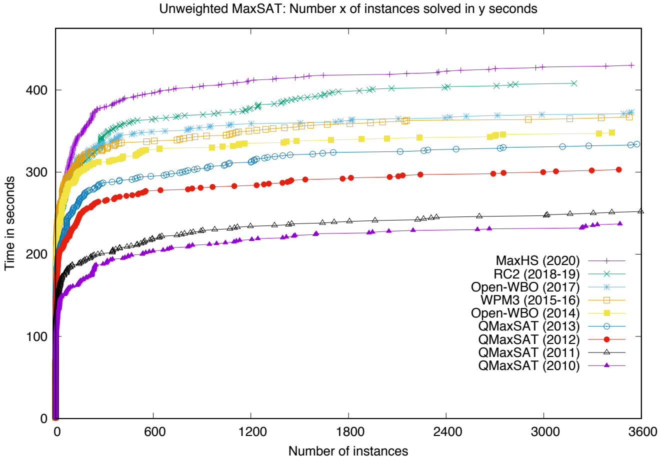
MaxSAT Applications

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*

This progress is much due to significant progress in efficient MaxSAT solvers.

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

Basic Concepts

MaxSAT: Basic Definitions

MaxSAT

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

TASK: find τ 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: τ 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)

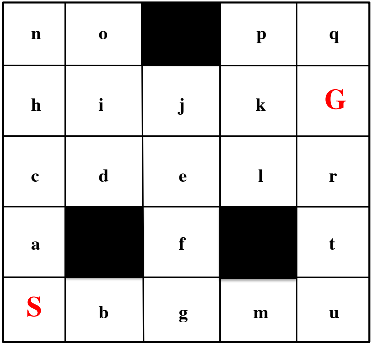
MaxSAT: Example

Shortest Path

Find shortest path in a grid with horizontal/vertical moves.

Travel from S to G.

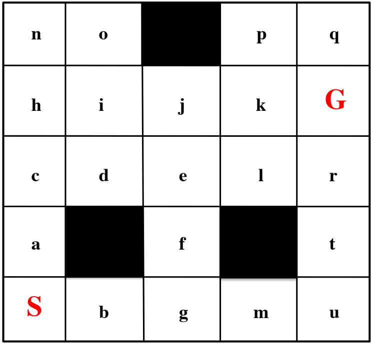
Cannot enter blocked squares.



MaxSAT: Example

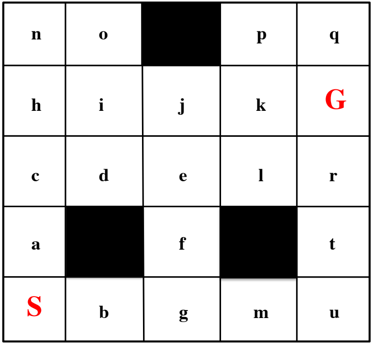
- Note: Best solved with state-space search
  - **Used here to illustrate how MaxSAT solving algorithms work and differ**

MAXSAT: Example



- Boolean variables: one for each unblocked grid square $\{S, G, a, b, \ldots, u\}$: 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: $(\neg a)$, $(\neg b)$, $\ldots$, $(\neg u)$ "*would prefer not to visit*"

MaxSAT: Example

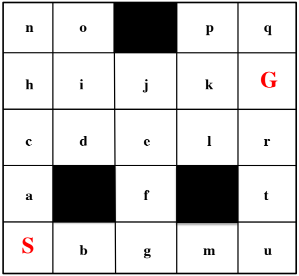
- Need to force the existence of a path between S and G by additional hard clauses

MaxSAT: Example

- 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:

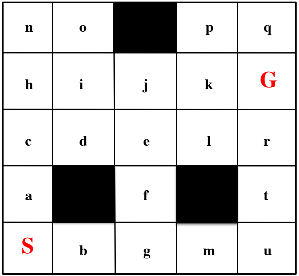
- both S and G must have *exactly one* visited neighbour
  - Any path starts from S
  - Any path ends at G
- other visited squares must have *exactly two* visited neighbours
  - One predecessor and one successor on the path



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: $(a \vee b), (\neg a \vee \neg b)$

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

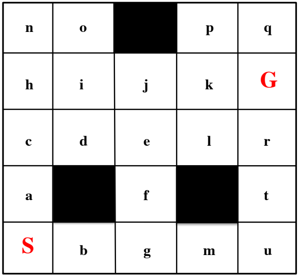
MaxSAT: Example

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

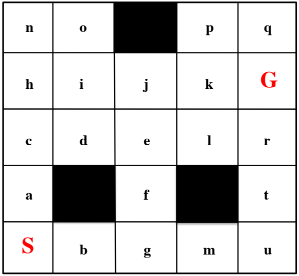


MaxSAT: Example

Constraint 2:

Other visited squares must have exactly two visited neighbours

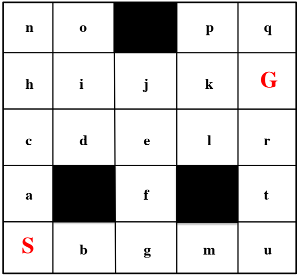
- For example, for square e: $e \rightarrow (d + j + l + f = 2)$



MaxSAT: Example

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

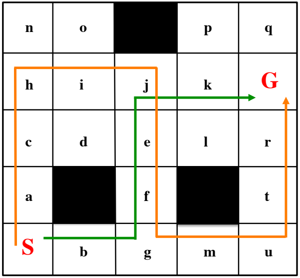
- For example, for square e: $e \rightarrow (d + j + l + f = 2)$
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF



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 $\mathsf{FP}^{\mathsf{NP}}$–complete

- Polynomial number of oracle calls
- A SAT solver acts as the NP oracle most often in practice

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
```

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
- *Various solvers from various research groups internationally* participate each year
- Standard input format
- Tracks for both *complete* and *incomplete* solvers

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

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

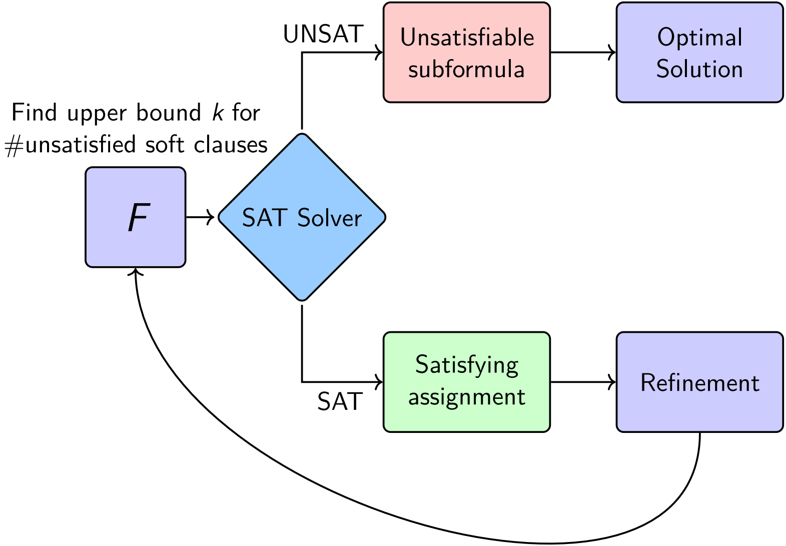
Types of Complete Solvers

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

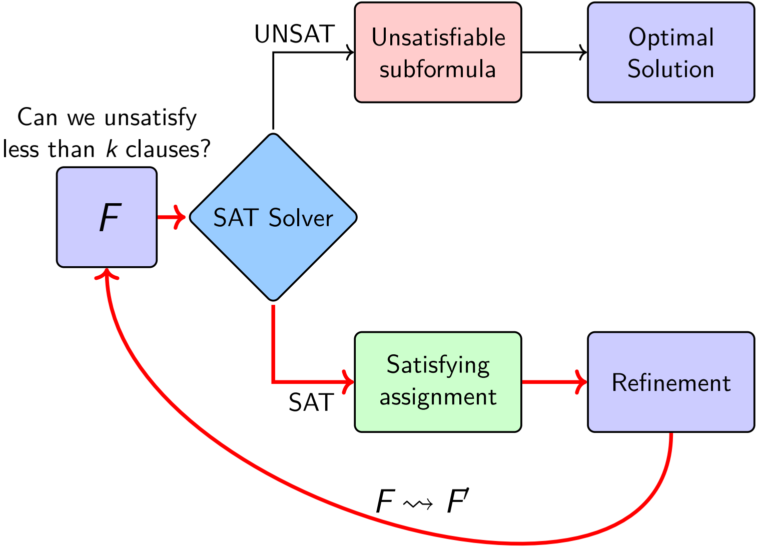
Focus here

- Complete solvers: Core-guided & implicit hitting set
- Incomplete: Combining different solving strategies

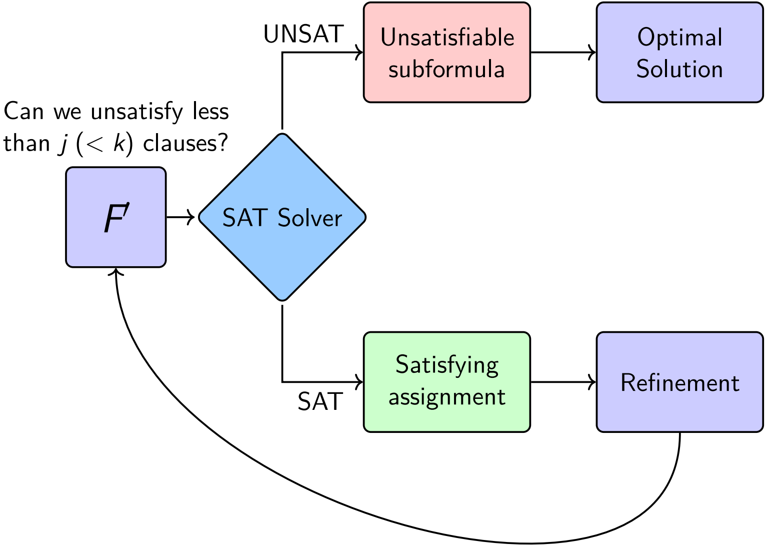
Model Improving: Upper Bound Search for MaxSAT



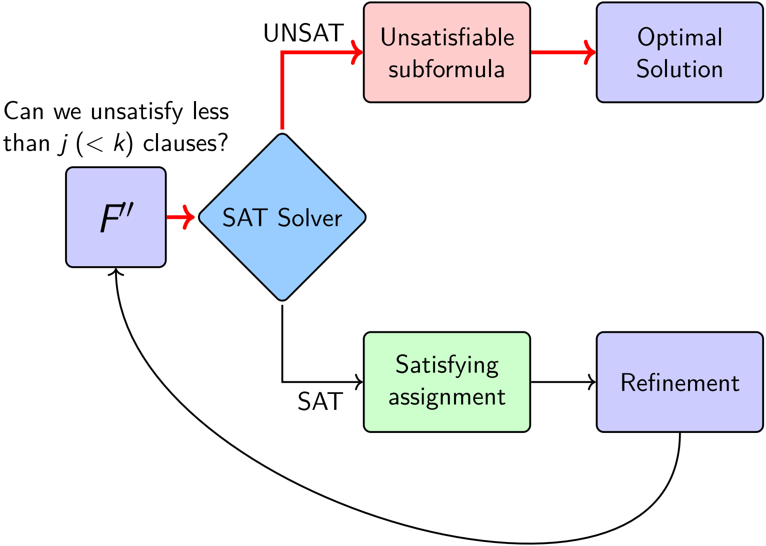
Model Improving: Upper Bound Search for MaxSAT



Model Improving: Upper Bound Search for MaxSAT



Model Improving: Upper Bound Search for MaxSAT



Model-Improving MaxSAT Solving

- 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

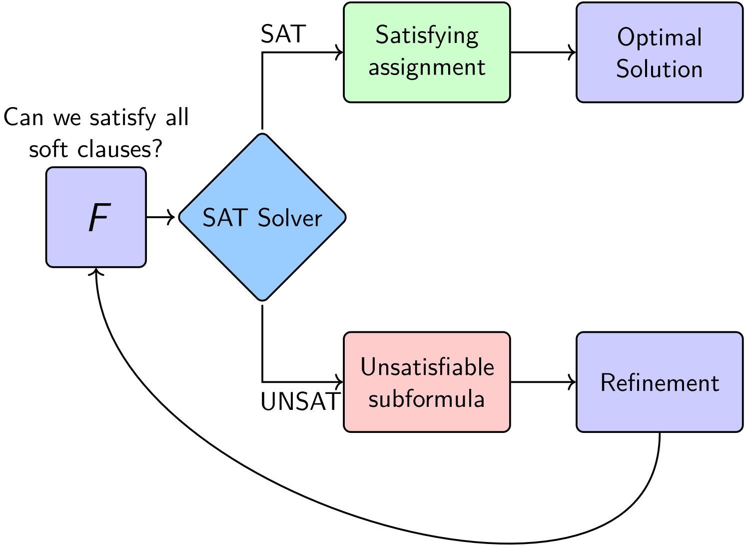
- Example of state-of-the-art solvers that use this algorithm:
  - QMaxSAT [Koshimura, Zhang, Fujita, and Hasegawa, 2012]
  - Pacose [Paxian, Reimer, and Becker, 2018]

- Also applied in *incomplete* MaxSAT solving — more on this later!

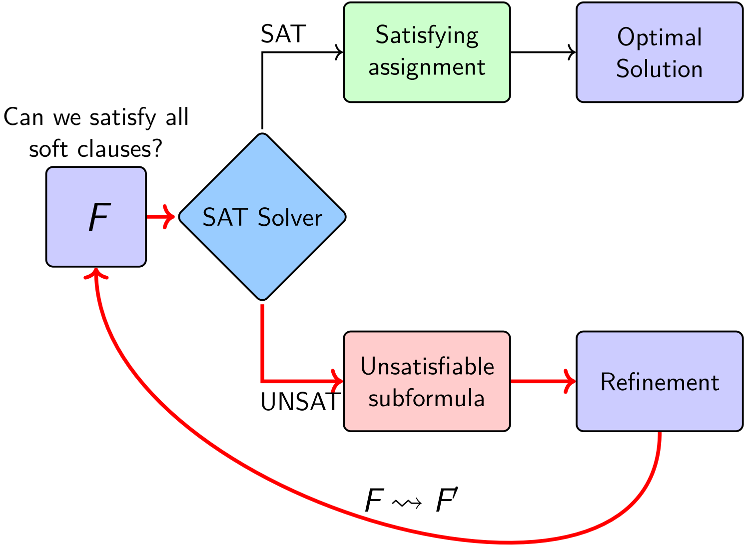
- Challenges:
  - Constraint that restricts the UB grows with the number of soft clauses (weights of the soft clauses)

Core-Guided MaxSAT Solving

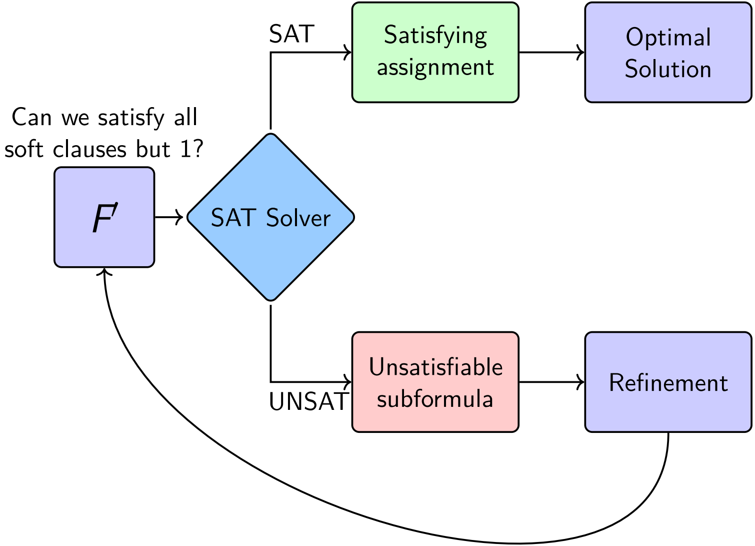
Lower Bound Search for MaxSAT



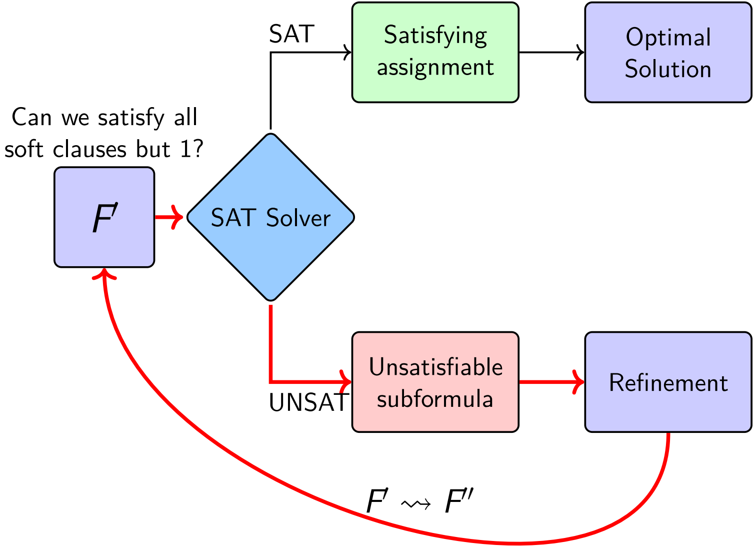
Lower Bound Search for MaxSAT



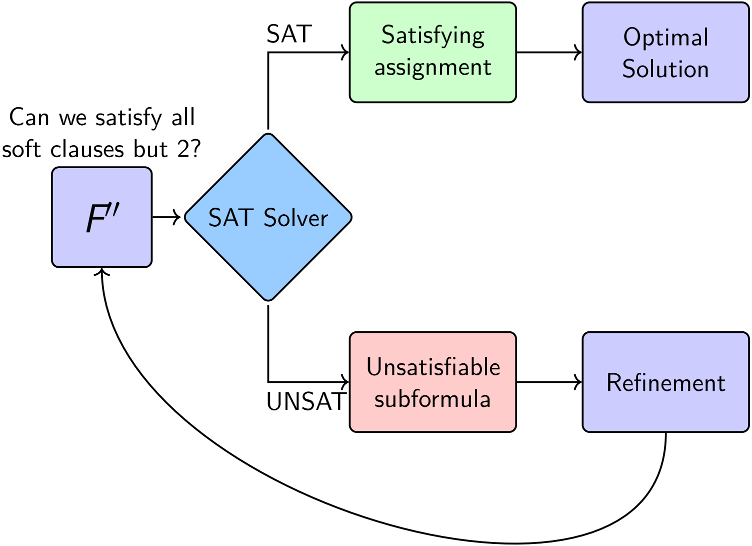
Lower Bound Search for MaxSAT



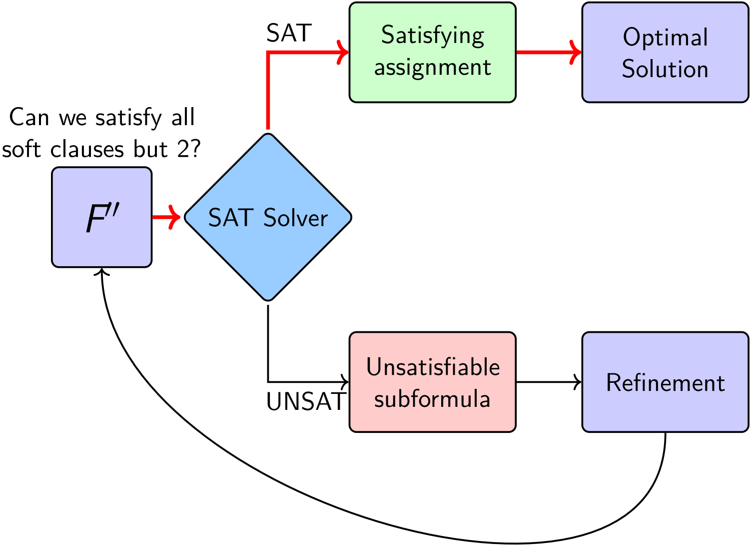
Lower Bound Search for MaxSAT



Lower Bound Search for MaxSAT



Lower Bound Search for MaxSAT



Unsatisfiability-based search for MaxSAT

- Simple idea:
  - For $LB = 0, \ldots$, query SAT solver on $H \wedge S \wedge \textsc{CostLessThan}(LB)$
  - Iterate until SAT solver reports satisfiable
  - The first model found will be optimal.

Unsatisfiability-based search for MaxSAT

- Simple idea:
  - For $LB = 0, \ldots$, query SAT solver on $H \wedge S \wedge \textsc{CostLessThan}(LB)$
  - Iterate until SAT solver reports satisfiable
  - The first model found will be optimal.
- 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 uses this algorithm

Unsatisfiability-based search for MaxSAT

- Simple idea:
  - For $LB = 0, \ldots$, query SAT solver on $H \wedge S \wedge \textsc{CostLessThan}(LB)$
  - Iterate until SAT solver reports satisfiable
  - The first model found will be optimal.
- 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 uses this algorithm
- 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

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 — UNSAT Cores

Formula:

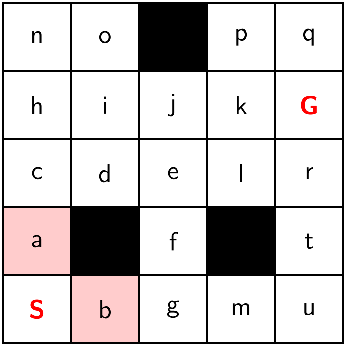
$$x_1 \quad x_3 \quad x_2 \vee \neg x_1 \quad \neg x_3 \vee x_1 \quad \neg x_2 \vee \neg x_1 \quad x_2 \vee \neg x_3$$

- Formula is unsatisfiable
- Unsatisfiable subformula (core):
  - $F' \subseteq F$, such that F' is unsatisfiable
    - **Subset of soft clauses** which together with the hard clauses constitute an unsatisfiable CNF formulas

Unsatisfiable subformula — UNSAT Cores

Formula:

- Formula is unsatisfiable
- Unsatisfiable subformula (core):
  - $F' \subseteq F$, such that F' is unsatisfiable
    - **Subset of soft clauses** which together with the hard clauses constitute an unsatisfiable CNF formulas



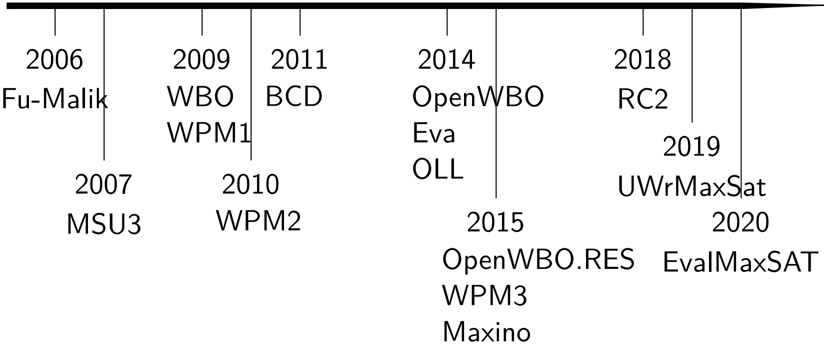
$$H = \{\ldots, (S), (S \rightarrow (a + b = 1), \ldots\}$$

$$\kappa = \{(\neg a), (\neg b)\} \subset S$$

$$\textsc{SAT-Solve}(H \wedge \kappa) = \text{UNSAT}$$

Core-Guided Algorithms

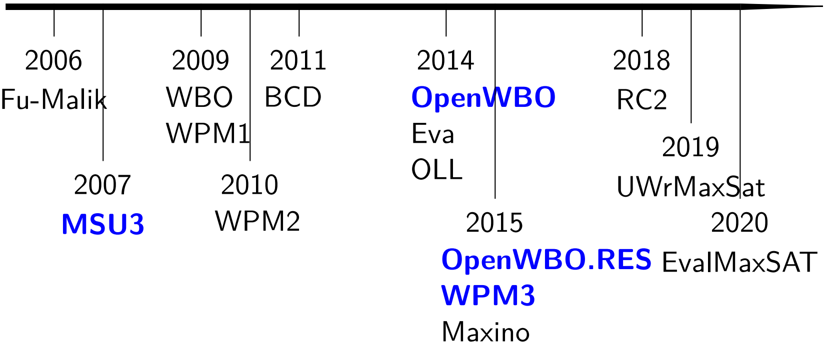
Timeline



- Various different core-guided solvers proposed.
- Focus here on a high-level view on how to use cores:

Core-Guided Algorithms

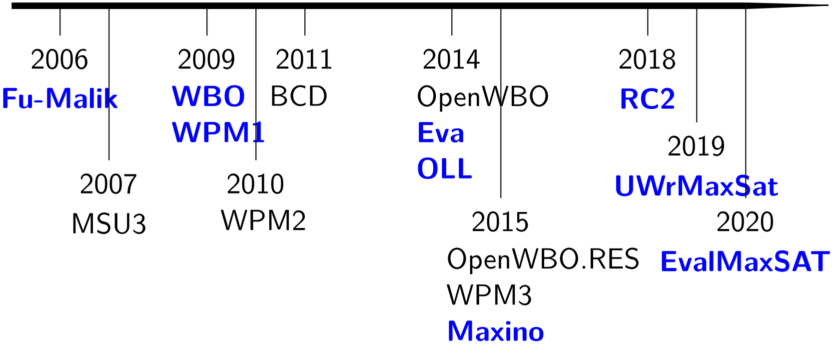
Timeline



- Various different core-guided solvers proposed.
- Focus here on a high-level view on how to use cores:
  - Relax cores together.

Core-Guided Algorithms

Timeline



- Various different core-guided solvers proposed.
- Focus here on a high-level view on how to use cores:
  - Relax cores together.
  - Relax cores separately.

Relax Cores together - MSU3

Shortest Path

Intuition

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

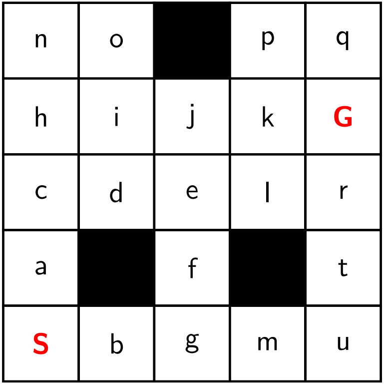
$\mathrm{LB} = 0, R = \{\}$

Relax Cores together - MSU3

Shortest Path

Intuition

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



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

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

is there a path that visits no nodes from the set R?

Relax Cores together - MSU3

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

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

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

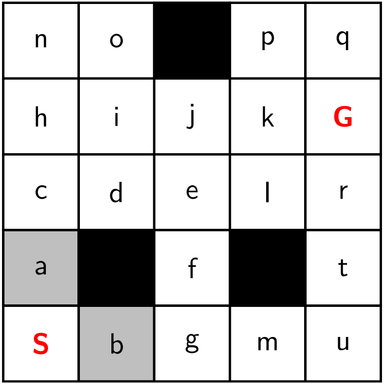
Formula is unsatisfiable
Use unsat core to update $R = \{a, b\}$

Relax Cores together - MSU3

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



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

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

is there a path that visits at most 1 nodes

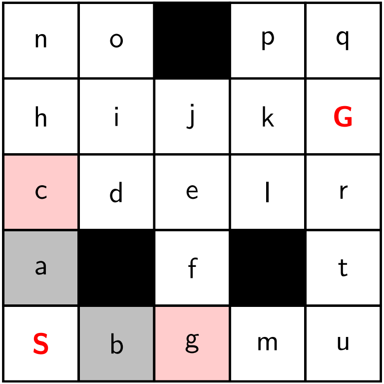
from the set R?

Relax Cores together - MSU3

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



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

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

Formula is unsatisfiable

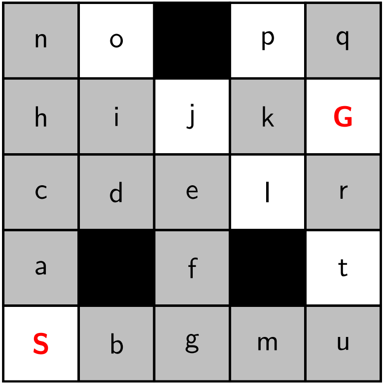
Use unsat core to update $R = \{a, b, c, g\}$

Relax Cores together - MSU3

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



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

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

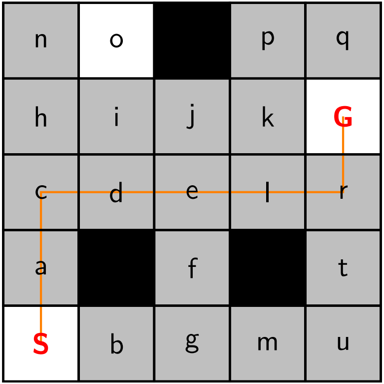
Formula is unsatisfiable

Relax Cores together - MSU3

Shortest Path

Intuition

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



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

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

$\tau = \{S, a, c, d, e, l, r, G, \neg b, \neg g, \ldots, \neg q\}$
$cost(\tau) = 6$

MSU3 Core-Guided Algorithm

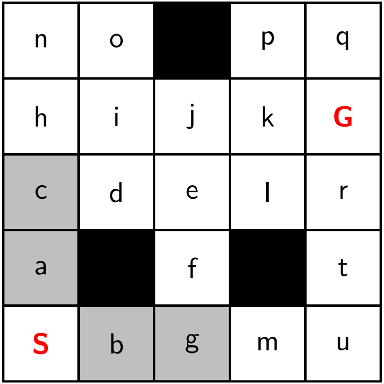
Summary

- 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, 2014]
- Challenges:
  - Constraint that restricts the LB grows with the size of cores
  - Does not capture local core information

MSU3

No Local Information:

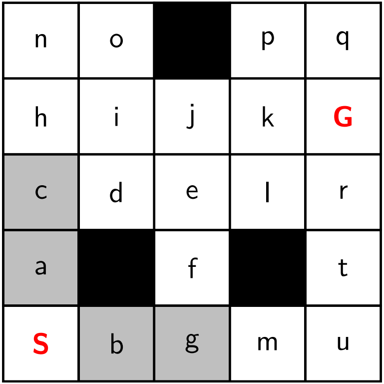
- In the third iteration, we are asking for a path that satisfies $a + b + c + g \leq 2$



MSU3

No Local Information:

- In the third iteration, we are asking for a path that satisfies $a + b + c + g \leq 2$
- Based on the cores we know $a + b \leq 1$ and $c + g \leq 1$



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

$\{(\neg a), (\neg b)\}$ *is a core*

$\rightarrow$ *all paths go through a or b*

$\{(\neg c), (\neg g)\}$ *is a core*

$\rightarrow$ *all paths go through c or g*

MSU3

No Local Information:

- In the third iteration, we are asking for a path that satisfies $a + b + c + g \leq 2$
- Based on the cores we know $a + b \leq 1$ **and** $c + g \leq 1$

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

Alternative:
Relax each core separately

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$

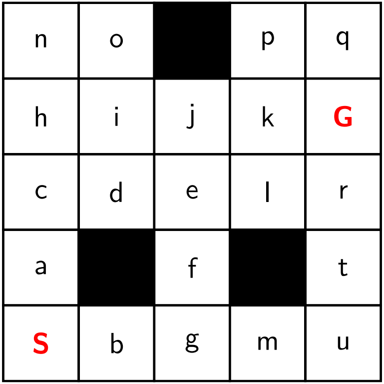
n	o		p	q
h	i	j	k	**G**
c	d	e	l	r
a		f		t
S	b	g	m	u

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable



$\mathrm{LB} = 0$, $i = 0$, $\mathcal{K} = \emptyset$

$\textsc{SAT-Solve}(\mathbf{H^i} \wedge \mathbf{S^i})$

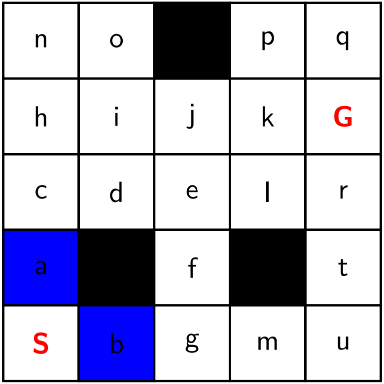
is there a path that visits at most 1 node from each found core

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i



$\mathrm{LB} = 0$, $i = 0$, $\mathcal{K} = \emptyset$

$\text{SAT-SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

Formula is unsatisfiable

Obtain new core: $\kappa_0 = \{(\neg a), (\neg b)\}$

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i

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

$\mathrm{LB} = 1,\ i = 1,\ \mathcal{K} = \{\kappa_0\}$

$\mathrm{SAT\text{-}SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

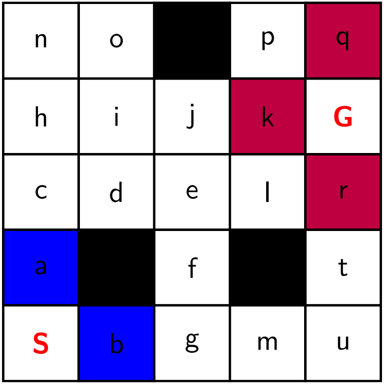
is there a path that visits at most 1 node from each found core

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i



$\mathrm{LB} = 1$, $i = 1$, $\mathcal{K} = \{\kappa_0\}$

$\mathrm{SAT\text{-}SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

Formula is unsatisfiable

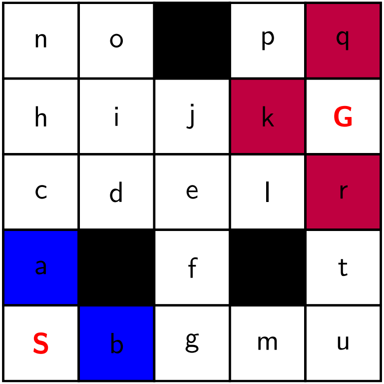
Obtain new core: $\kappa_1 = \{(\neg q), (\neg k), (\neg r)\}$

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i



$\mathrm{LB} = 2$, $i = 2$, $\mathcal{K} = \{\kappa_0, \kappa_1\}$

$\textsc{Sat-Solve}(\mathbf{H^i} \wedge \mathbf{S^i})$

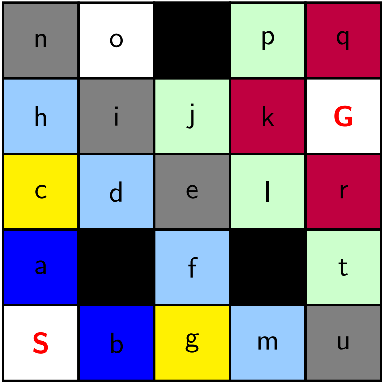
is there a path that visits at most 1 node from each found core

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i



$\mathrm{LB} = 6$, $i = 6$, $\mathcal{K} = \{\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5\}$

$\mathrm{SAT\text{-}SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

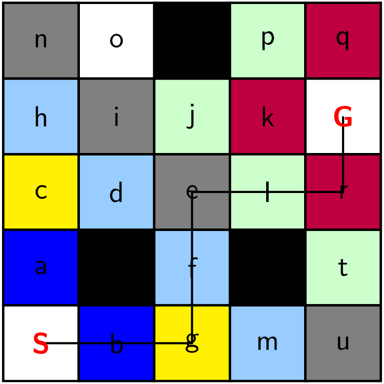
is there a path that visits at most 1 node from each found core

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i
4. Otherwise, the obtained model is optimal



$\mathrm{LB} = 6$, $i = 6$, $\mathcal{K} = \{\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5\}$

$\mathrm{SAT\text{-}SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

Formula is satisfiable

Obtain optimal model: $\tau = \{b, \ldots, l, r, \neg a, \ldots, \neg q\}$

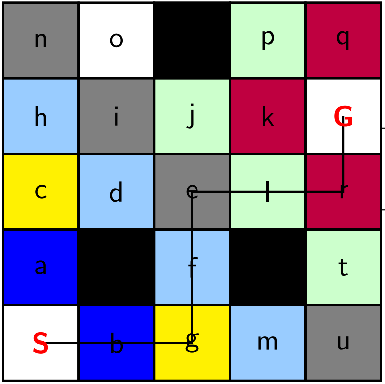
$cost(\tau) = 6$

Relaxing Cores Separately

Shortest path

Intuition

1. Initialise $H^0 = H$ and $S^0 = S$
2. For $i = 0, \ldots$ check if $H^i \wedge S^i$ is satisfiable
3. If not, relax H^i and S^i
4. Otherwise, the obtained model is optimal



$\mathrm{LB} = 6$, $i = 6$, $\mathcal{K} = \{\kappa_0, \kappa_1, \kappa_2, \kappa_3, \kappa_4, \kappa_5\}$

Note:
LB and $\mathcal{K}$ maintained only implicitly

Separate Core relaxation

Summary

- First instantiated by Fu-Malik [Fu and Malik, 2006]
- Other early instantiations in WBO, and WPM1 [Manquinho, Marques-Silva, and Planes, 2009; Ansótegui, Bonet, and Levy, 2009]
- Today, most solvers use OLL [Andres, Kaufmann, Matheis, and Schaub, 2012; Morgado, Dodaro, and Marques-Silva, 2014]

Separate Core relaxation

Summary

- First instantiated by Fu-Malik [Fu and Malik, 2006]
- Other early instantiations in WBO, and WPM1 [Manquinho, Marques-Silva, and Planes, 2009; Ansótegui, Bonet, and Levy, 2009]
- Today, most solvers use OLL [Andres, Kaufmann, Matheis, and Schaub, 2012; Morgado, Dodaro, and Marques-Silva, 2014]
- Benefits:
  - Encoding cardinality constraints into CNF is efficient since it only uses AtMost 1 constraints

Separate Core relaxation

Summary

- First instantiated by Fu-Malik [Fu and Malik, 2006]
- Other early instantiations in WBO, and WPM1 [Manquinho, Marques-Silva, and Planes, 2009; Ansótegui, Bonet, and Levy, 2009]
- Today, most solvers use OLL [Andres, Kaufmann, Matheis, and Schaub, 2012; Morgado, Dodaro, and Marques-Silva, 2014]
- Benefits:
  - Encoding cardinality constraints into CNF is efficient since it only uses AtMost 1 constraints
- Challenges:
  - Multiple cardinality constraints
  - Extracting cores from reformulated formula can be exponentially harder [Bacchus and Narodytska, 2014]

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 $\textsc{MaxSAT}$?

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

Hitting Sets and UNSAT Cores

Key insight

To find an optimal solution to a $\textsc{MaxSAT}$ 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

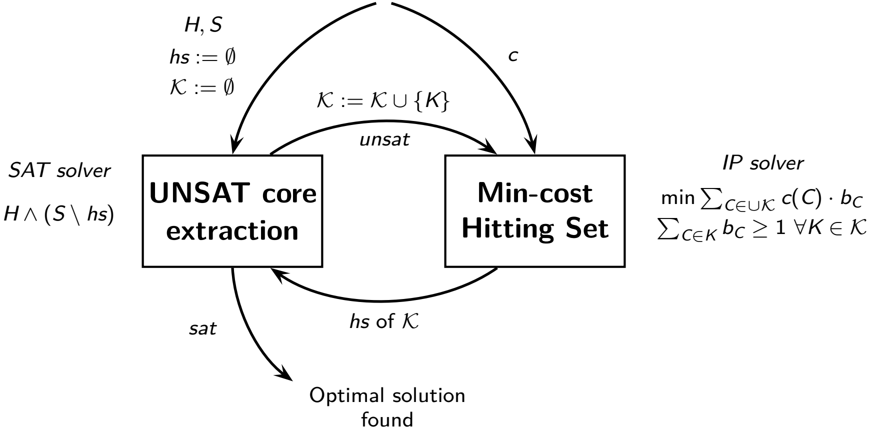
"Best out of both worlds"

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

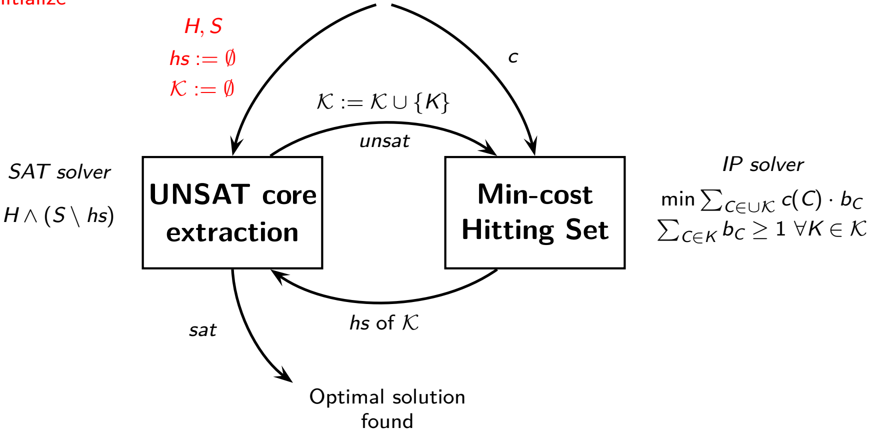
Input:
hard clauses H, soft clauses S, weight function $c : S \mapsto \mathbb{R}^+$



Solving MaxSAT by SAT and Hitting Set Computations

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

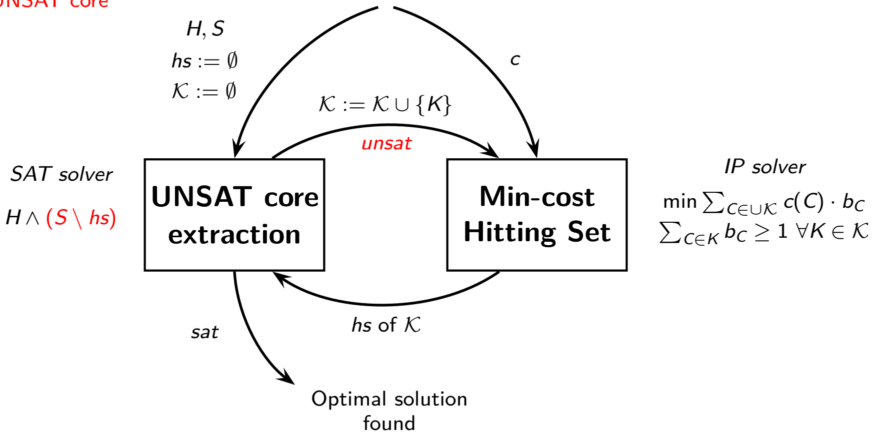
1. Initialize



Solving MaxSAT by SAT and Hitting Set Computations

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

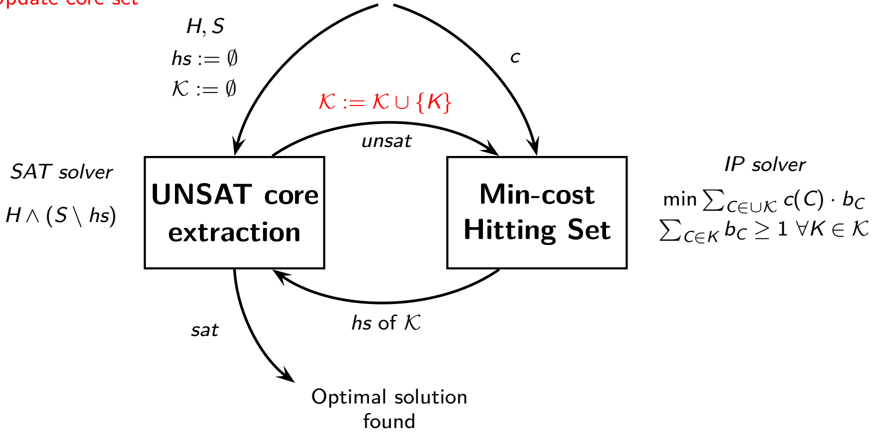
2. UNSAT core



Solving MaxSAT by SAT and Hitting Set Computations

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

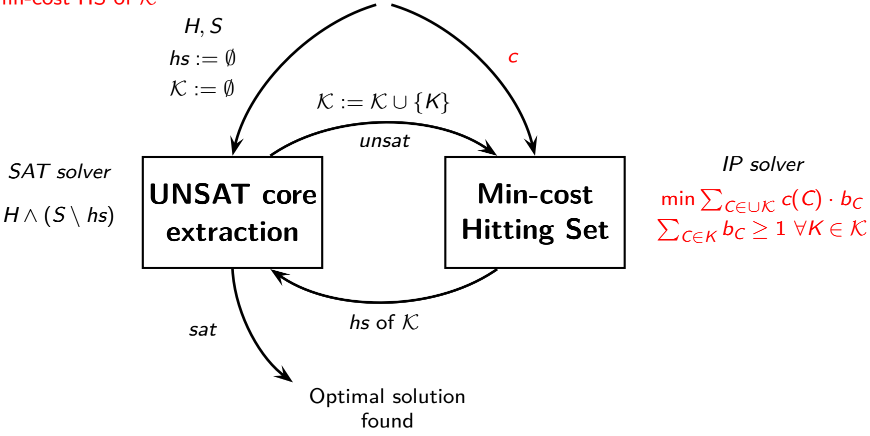
3. Update core set



Solving MAXSAT by SAT and Hitting Set Computations

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

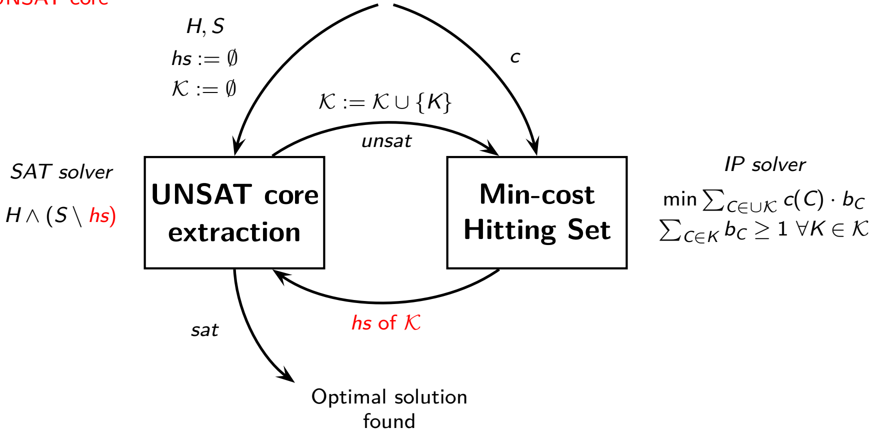
4. Min-cost HS of $\mathcal{K}$



Solving MaxSAT by SAT and Hitting Set Computations

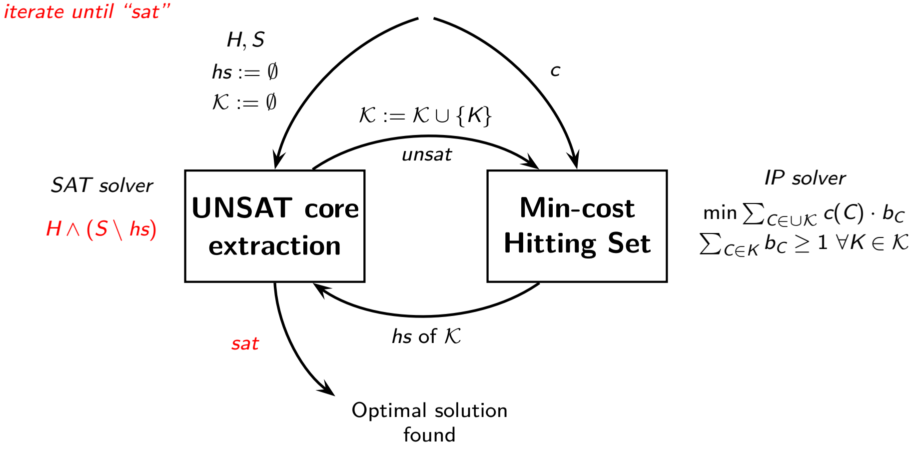
Input:
hard clauses H, soft clauses S, weight function $c : S \mapsto \mathbb{R}^+$

5. UNSAT core



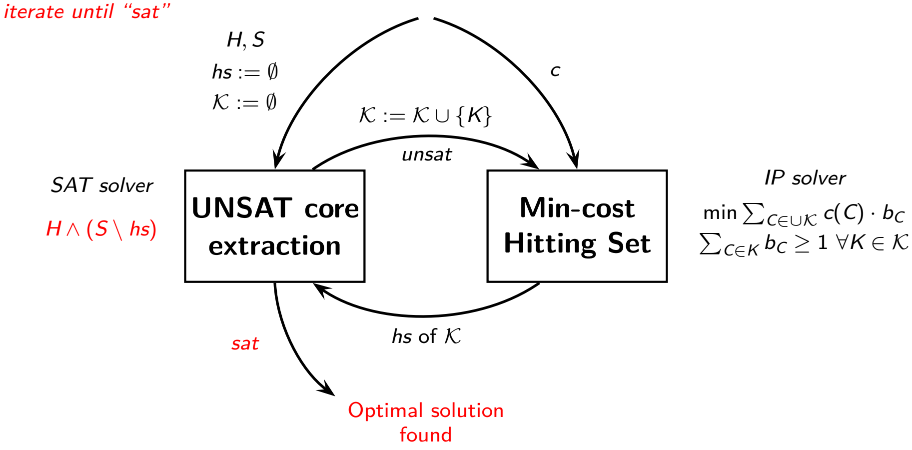
Solving MaxSAT by SAT and Hitting Set Computations

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



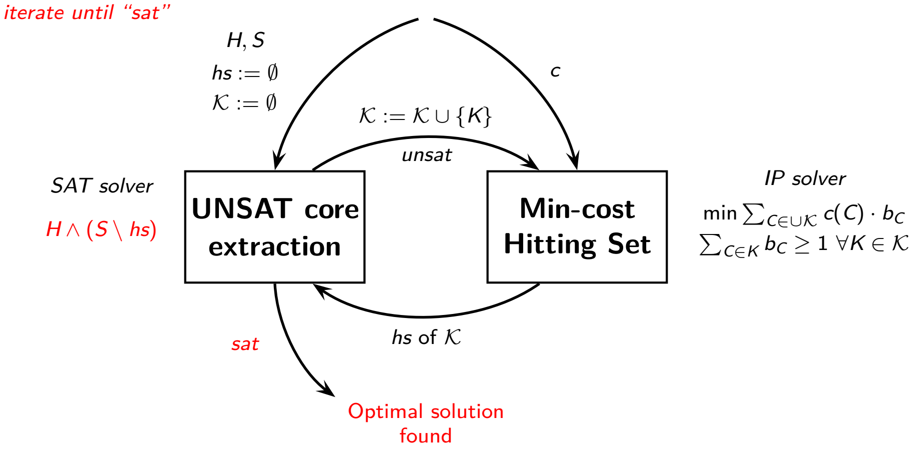
Solving MaxSAT by SAT and Hitting Set Computations

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.



Intuition

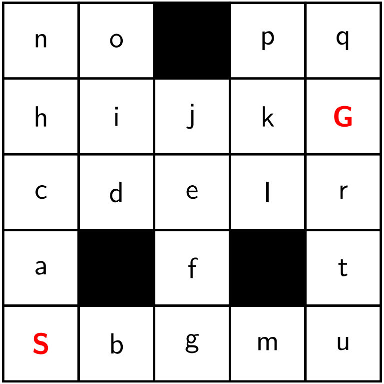
Hitting sets provide candidate paths.
Sat solver verifies the candidates.

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



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

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.

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

$\mathcal{K} = \emptyset$

$hs = \emptyset$

$\textbf{IP-solve}(\mathcal{K}) = \emptyset$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.

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

$\mathcal{K} = \emptyset$

$hs = \emptyset$

SAT-Solve$(H \wedge (S \setminus hs))$

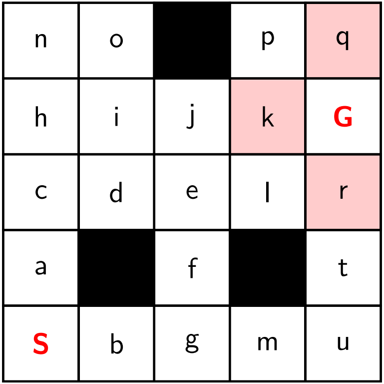
is there a path via no nodes?

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\{(\neg q), (\neg k), (\neg r)\}\}$

$hs = \emptyset$

$\textsc{SAT-Solve}(H \wedge (S \setminus hs))$

is there a path via no nodes?

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

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

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.

Sat solver verifies the candidates.

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

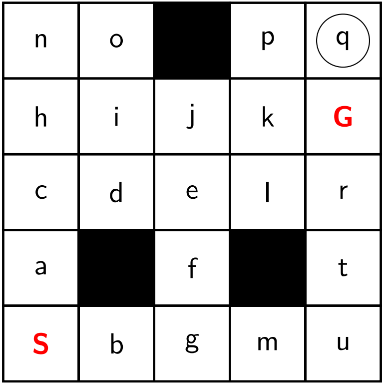
$\mathcal{K} = \{\{(\neg q), (\neg k), (\neg r)\}\}$

$hs = \{(\neg q)\}$

$\textbf{IP-solve}(\mathcal{K}) = \{(\neg q)\}$

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\{(\neg q), (\neg k), (\neg r)\}\}$
$hs = \{(\neg q)\}$

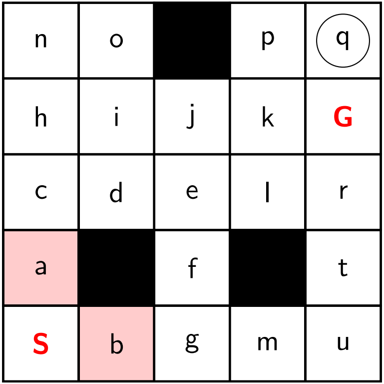
SAT-Solve$(H \wedge (S \setminus hs))$
is there a path that only goes through q?

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\{(\neg q), (\neg k), (\neg r)\}, \{(\neg a), (\neg b)\}\}$
$hs = \{(\neg q)\}$

$\text{SAT-Solve}(H \wedge (S \setminus hs))$
is there a path that only goes through q?

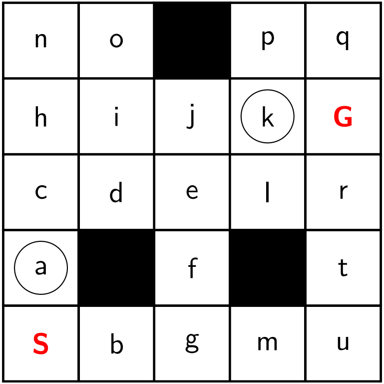
Result: UNSAT Core: $\{(\neg a), (\neg b)\}$
i.e. all paths go through a or b

$\textsc{MaxSAT}$ by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



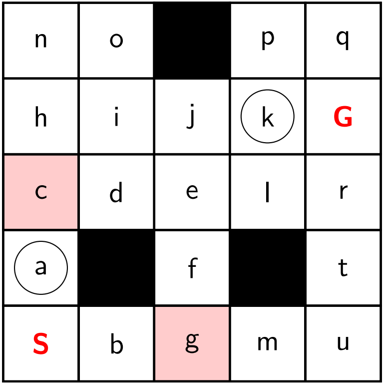
$\mathcal{K} = \{\{(\neg q), (\neg k), (\neg r)\}, \{(\neg a), (\neg b)\}\}$

$hs = \{(\neg k), (\neg a)\}$

IP-solve$(\mathcal{K}) = \{(\neg k), (\neg a)\}$

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg k), (\neg a)\}$

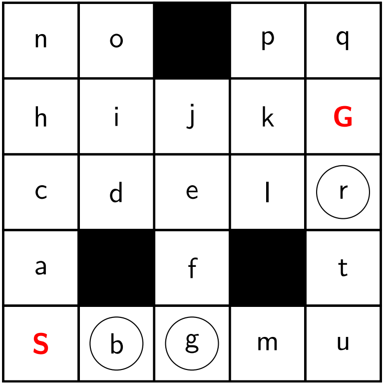
$\textsc{SAT-Solve}(H \wedge (S \setminus hs)) = \text{UNSAT}$
Core: $\{(\neg c), (\neg g)\}$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg b), (\neg g), (\neg r)\}$

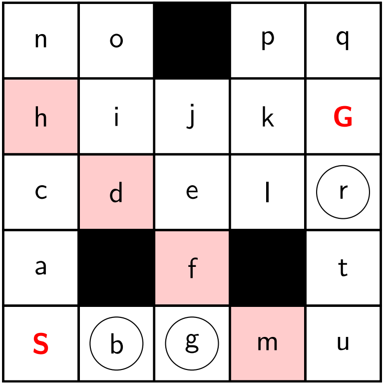
$\text{IP-SOLVE}(\mathcal{K}) = \{(\neg b), (\neg g), (\neg r)\}$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg b), (\neg g), (\neg r)\}$

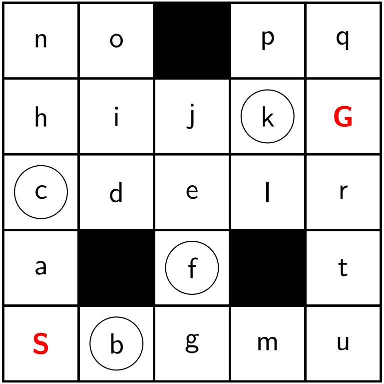
$\textsc{SAT-Solve}(H \wedge (S \setminus hs)) = \text{UNSAT}$
Core: $\{(\neg h), (\neg d), (\neg f), (\neg m)\}$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg b), (\neg c), (\neg f), (\neg k)\}$

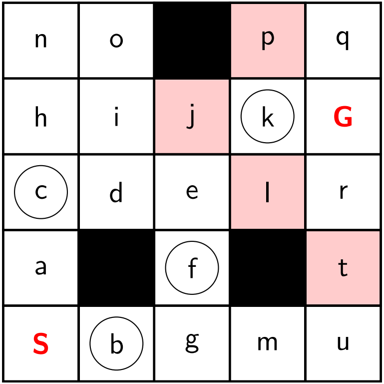
$\text{IP-SOLVE}(\mathcal{K}) = \{(\neg b), (\neg c), (\neg f).(\neg f)\}$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg b), (\neg c), (\neg f), (\neg k)\}$

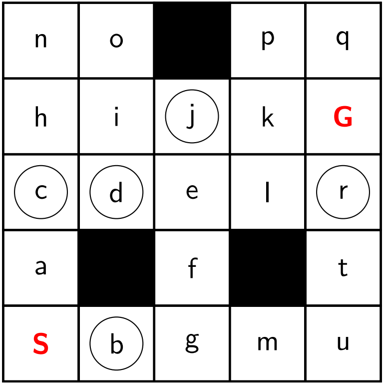
$\textsc{SAT-Solve}(H \wedge (S \setminus hs)) = \text{UNSAT}$
Core: $\{(\neg p), (\neg j), (\neg l), (\neg t)\}$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$

$hs = \{(\neg b), (\neg c), (\neg d), (\neg j), (\neg r)\}$

$\text{IP-SOLVE}(\mathcal{K}) = \{(\neg b), (\neg c), (\neg d), (\neg j), (\neg r)\}$

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.

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

$\mathcal{K} = \{\kappa_1, \ldots\}$
$hs = \{(\neg b), (\neg c), (\neg d), (\neg j), (\neg r)\}$

$\text{SAT-SOLVE}(H \wedge (S \setminus hs)) = \text{UNSAT}$
Core: $\{(\neg n), (\neg i), (\neg e), (\neg u)\}$

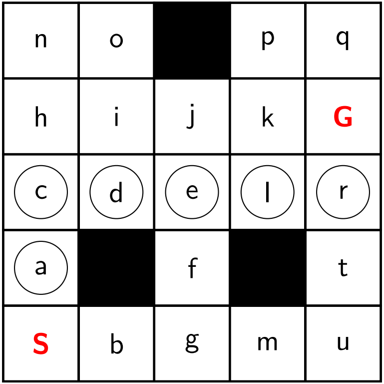
MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.

Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$

$hs = \{(\neg a), (\neg c), (\neg d), (\neg e), (\neg l), (\neg r)\}$

$\textsc{IP-solve}(\mathcal{K}) =$

$\{(\neg a), (\neg c), (\neg d), (\neg e), (\neg l), (\neg r)\}$

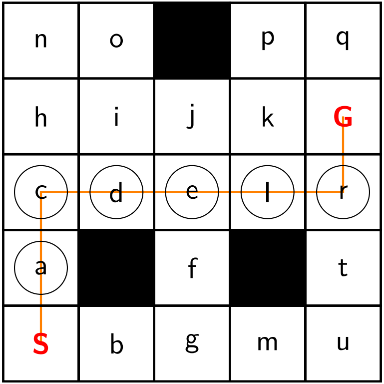
MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.

Sat solver verifies the candidates.



$\mathcal{K} = \{\kappa_1, \ldots\}$

$hs = \{(\neg a), (\neg c), (\neg d), (\neg e), (\neg l), (\neg r)\}$

$\textsc{SAT-Solve}(H \wedge (S \setminus hs))$

is there a path that through a, c, d, e, l, r?

Result: SAT

$\tau = \{a, c, \ldots, r, \neg b, \neg h, \ldots, \neg q\}$

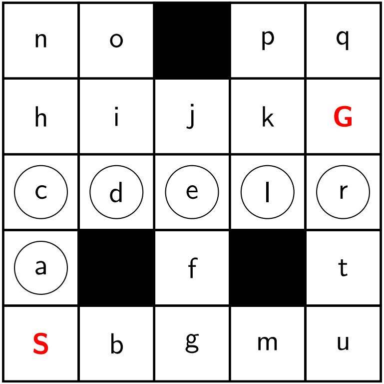
$cost(\tau) = 6$

MaxSAT by SAT and Hitting Set Computation

Shortest Path

Intuition

Hitting sets provide candidate paths.
Sat solver verifies the candidates.



Note:
Termination requires computing the correct hitting set

$hs = \{(\neg a), (\neg c), (\neg d), (\neg e), (\neg l), (\neg r)\}$

$hs = \{(\neg a), (\neg g), (\neg f), (\neg i), (\neg p), (\neg q)\}$

$hs = \{(\neg b), (\neg c), (\neg m), (\neg i), (\neg j), (\neg r)\}$

$\vdots$

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.
- 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.
- 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?

(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.

Approaches to Incomplete MaxSAT

Model-Improving Search

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

complete & any-time

Core-Boosted search

Combine core-guided and model-improving search.

complete & any-time

Stochastic Local Search (SLS)

Quickly traverse the search space by local changes to current solution

Improved with a SAT solver

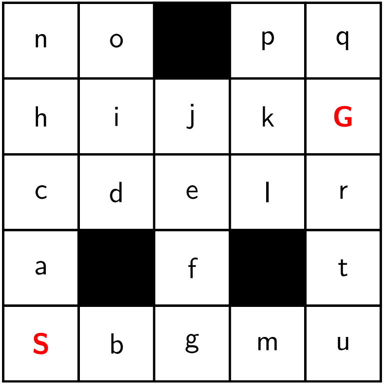
incomplete

Model-Improving Search

Model-Improving Algorithm

Shortest Path

Intuition



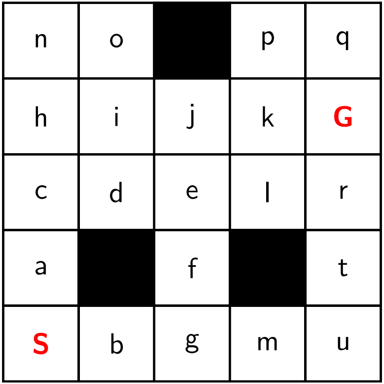
$\mathrm{UB} = \infty$

Model-Improving Algorithm

Shortest Path

Intuition

1. Obtain a solution τ^*



$\mathrm{UB} = \infty$

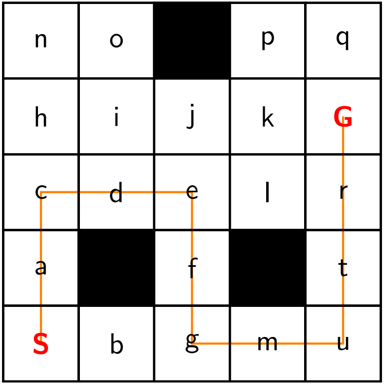
SAT-SOLVE(**H**)

Model-Improving Algorithm

Shortest Path

Intuition

1. Obtain a solution τ^*
2. Update UB



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

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 τ^*
2. Update UB
3. Improve τ^* until τ^* is proven to be optimal

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

$\mathrm{UB} = 10$

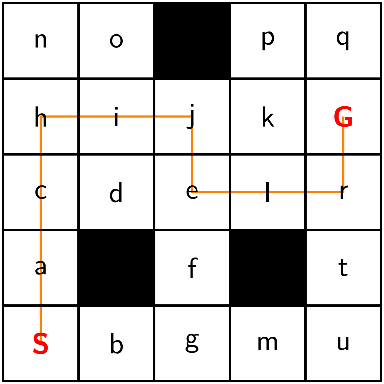
$\textsc{SAT-SOLVE}\,(\mathbf{H} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{UB}))$

Model-Improving Algorithm

Shortest Path

Intuition

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



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

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

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

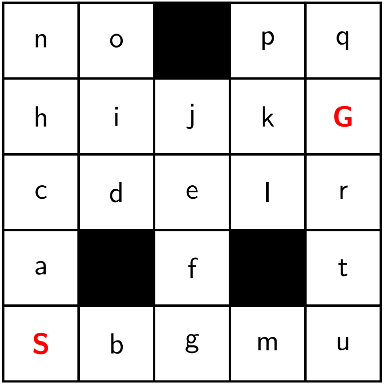
$cost(\tau^2) = 8$

Model-Improving Algorithm

Shortest Path

Intuition

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



$\mathrm{UB} = 8$

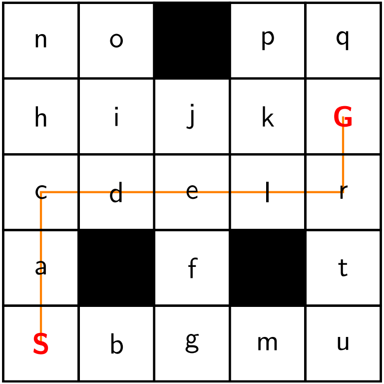
$\text{SAT-SOLVE}\,(\mathbf{H} \wedge \textsc{CostLessThan}(\mathbf{S}, \mathbf{UB}))$

Model-Improving Algorithm

Shortest Path

Intuition

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



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

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

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

$cost(\tau^3) = 6$

Model-Improving 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:
  - Rescale weights.

$$S = \{(C_1, 100), (C_2, 101), (C_3, 123), (C_4, 1205), (C_5, 1540), (C_6, 1260) \ldots$$

Divide by 100

$$S = \{(C_1, 1), (C_2, 1), (C_3, 1), (C_4, 12), (C_5, 15), (C_6, 12) \ldots\}$$

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:
  - Rescale weights.

$$S = \{(C_1, 100), (C_2, 101), (C_3, 123), (C_4, 1205), (C_5, 1540), (C_6, 1260) \ldots$$

Divide by 500

$$S = \{(C_4, 2), (C_5, 3), (C_6, 2) \ldots\}$$

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:
  - Rescale weights.
  - Core-Boosted Search

Core-Boosted Search

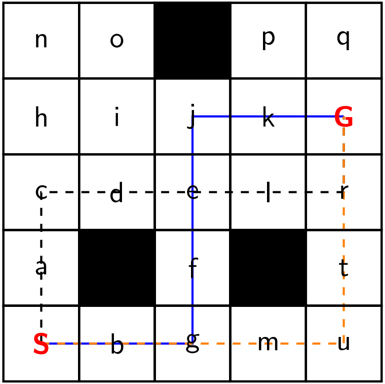
Core-Boosted Search

Shortest path

[Berg, Demirovic, and Stuckey, 2019]

Intuition

1. Solutions to $F = (H^0, S^0) \rightarrow$ shortest paths from S to G



$\mathrm{LB} = 0,\ i = 0,\ \mathcal{K} = \emptyset$

$\mathrm{MAXSAT\text{-}SOLVE}(\mathbf{H^i} \wedge \mathbf{S^i})$

$cost(F) = 6$

What is the length of the shortest path from S to G?

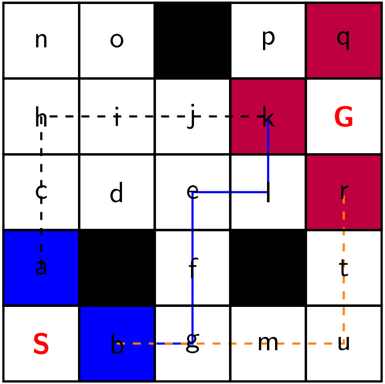
Core-Boosted Search

Shortest path

[Berg, Demirovic, and Stuckey, 2019]

Intuition

1. Solutions to $F = (H^0, S^0) \rightarrow$ shortest paths from S to G
2. Solutions to $F^2 = (H^2, S^2) \rightarrow$ shortest paths from κ_0 to κ_1



$\mathrm{LB} = 2$, $i = 2$, $\mathcal{K} = \{\kappa_0, \kappa_1\}$

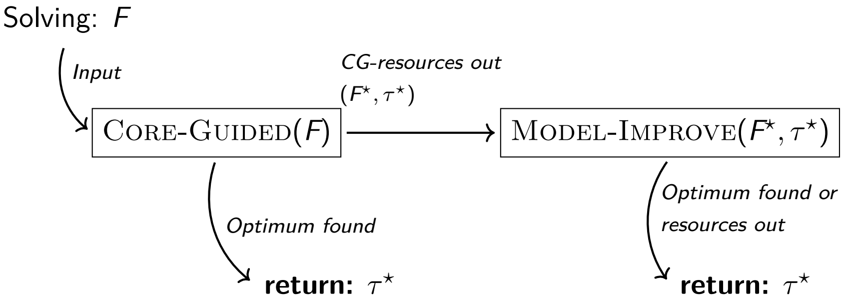
MAXSAT-SOLVE($\mathbf{H^i} \wedge \mathbf{S^i}$)

$cost(F^2) = 4$

What is the length of the shortest path from a or b to q, k, or r?

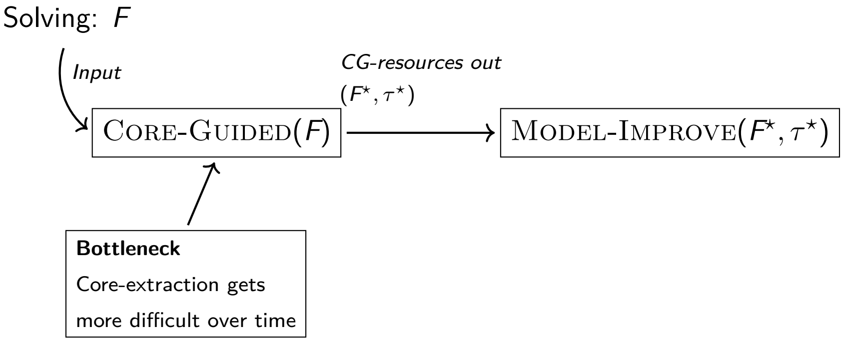
Core-Boosted Linear Search

In General



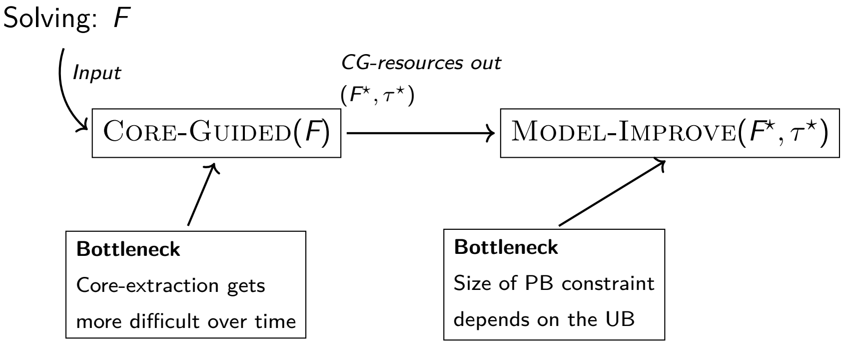
Core-Boosted Linear Search

In General



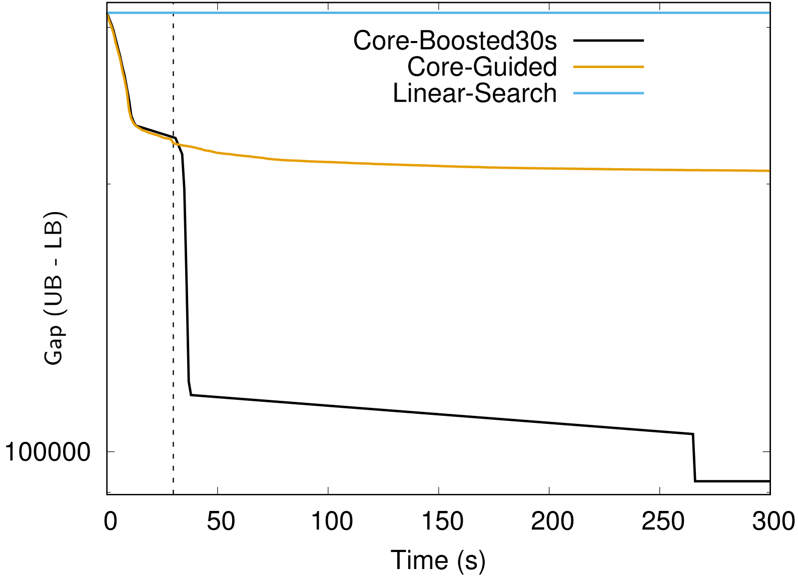
Core-Boosted Linear Search

In General



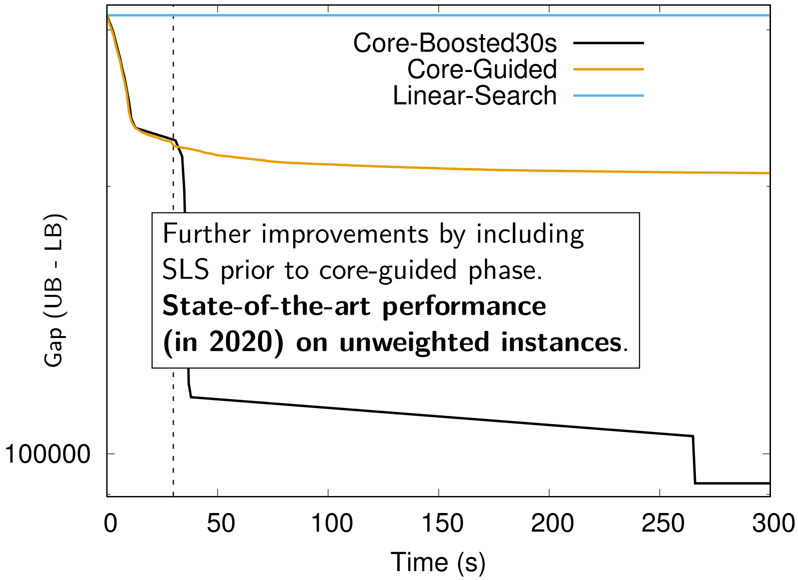
Core-Boosted search

Example



Core-Boosted search

Example



Stochastic Local Search for MaxSAT

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:

- Extend weights to all clauses

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.

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:

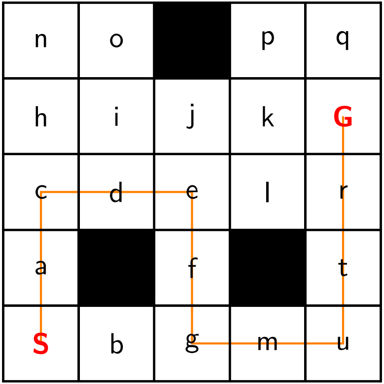
- Extend weights to all clauses
  - Initialize weight of all hard clauses to 1
  - Flip literals from unsatisfied clauses with high weight.
  - Periodically increase weights of clauses that are frequently unsatisfied.
- Use a SAT solver to satisfy the hard clauses

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*



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

$\mathrm{UB} = 10$

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

$cost(\tau^*) = 10$

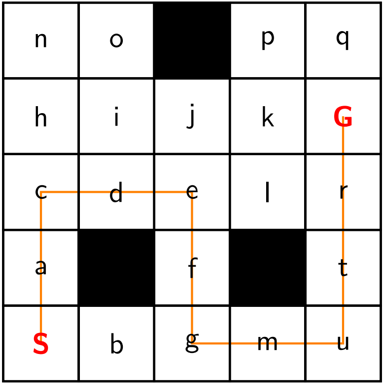
$\mathrm{FIXED} = \emptyset$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

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



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

$\mathrm{UB} = 10$

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

$cost(\tau^*) = 10$

$\mathrm{FIXED} = \emptyset$

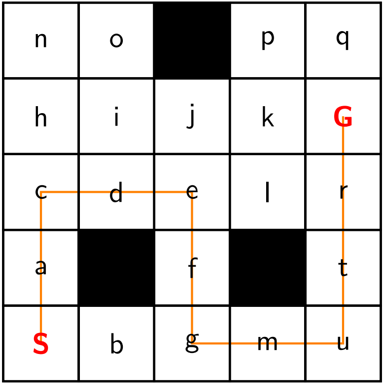
Current: $(\neg a)$ $\quad$ $\tau^*(\neg a) = 0$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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: $(\neg a)$ $\quad$ $\tau^*(\neg a) = 0$

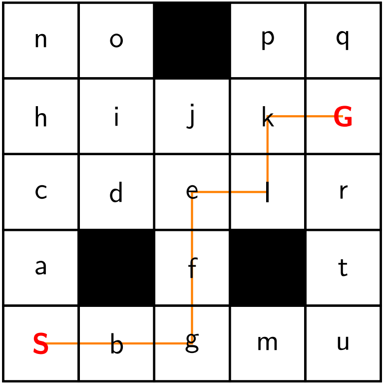
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg a))$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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: $(\neg a)$ $\tau^*(\neg a) = 0$

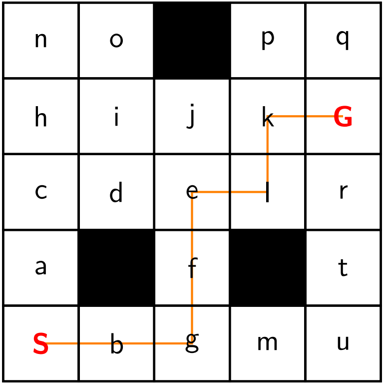
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg a))$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

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



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

$\mathrm{UB} = 6$

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

$cost(\tau^*) = 6$

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

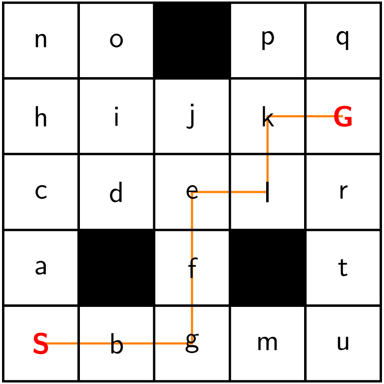
Current: $(\neg b)$ $\tau^*(\neg b) = 0$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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: $(\neg b)$ $\tau^*(\neg b) = 0$

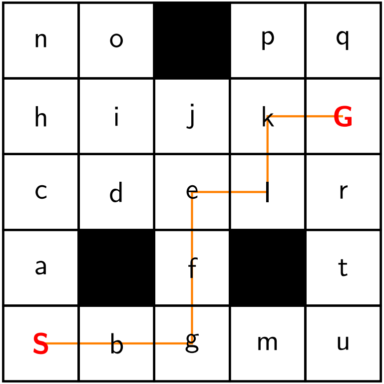
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg b))$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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: $(\neg b)$ $\tau^*(\neg b) = 0$

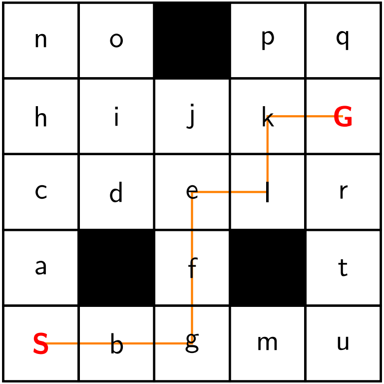
$\mathrm{SATSOLVE}(H \wedge \bigwedge_{C \in \mathrm{FIXED}} C \wedge (\neg b))$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

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



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

$\mathrm{UB} = 6$

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

$cost(\tau^*) = 6$

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

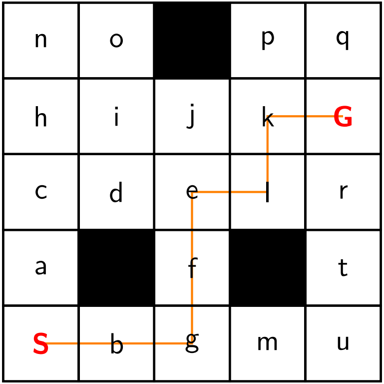
Current: $(\neg c)$ $\tau^*(\neg c) = 0$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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: $(\neg c)$ $\tau^*(\neg c) = 0$

SLS with a SAT solver

Nadel [2018, 2019]

Intuition

1. Obtain any solution τ^*
2. Improve τ^* 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

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.
Longer answer (in 2021): Try TT-Open-WBO-Inc for weighted and SATLike (2020 version) or Loandra for unweighted.

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

Further Reading and Links

Talks at the Simons Institute

- Fahiem Bacchus on (complete) MaxSAT on April 1st.
- Jeremias on MaxSAT preprocessing on May 5th.

Surveys

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

    [Ansótegui, Bonet, and Levy, 2013]

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

MaxSAT Evaluations

https://maxsat-evaluations.github.io

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

Thank you for attending!

Bibliography I

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. SAT-based MaxSAT algorithms. *Artificial Intelligence*, 196: 77–105, 2013. doi: 10.1016/j.artint.2013.01.002. URL `http://dx.doi.org/10.1016/j.artint.2013.01.002`.

Fahiem Bacchus and Nina Narodytska. Cores in core based MaxSAT algorithms: An analysis. 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 7–15. Springer, 2014.

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`.

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`.

Bibliography II

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`.

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`.

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.

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`.

Bibliography III

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.

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`.

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`.

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`.

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`.

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, 2014.

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, 2013. doi: 10.1007/s10601-013-9146-2. URL `http://dx.doi.org/10.1007/s10601-013-9146-2`.

Bibliography IV

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`.

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`.

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.

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.