Algorithms for Maximum Satisfiability

with Applications to AI

Fahiem Bacchus[1] Matti Järvisalo[2]

[1]University of Toronto
Canada

[2]University of Helsinki
Finland

February 13, 2016 AAAI-16, Phoenix, AZ, USA

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 “very Boolean” optimization problems

Tutorial Outline

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

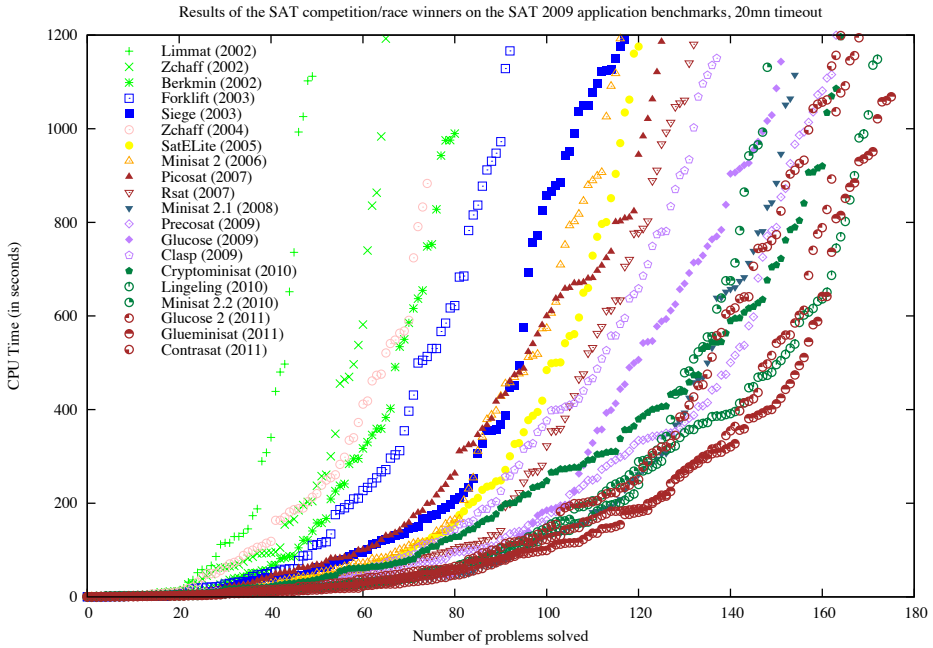
Not merely a central problem in *theory*:

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

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

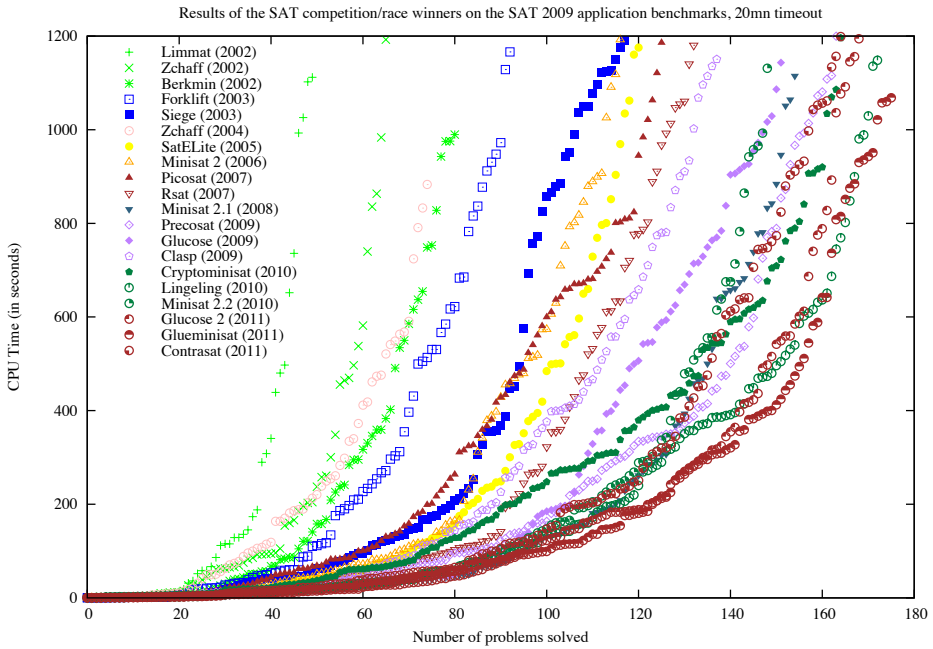
SAT Solvers

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



SAT Solvers

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



Core NP search procedures for solving various types of computational problems

Exact Optimization

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

Importance of Exact Optimization

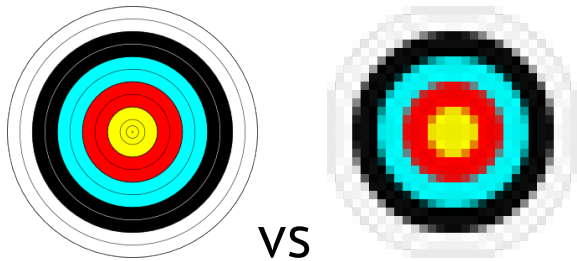
Giving Up?

"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



Importance of Exact Optimization

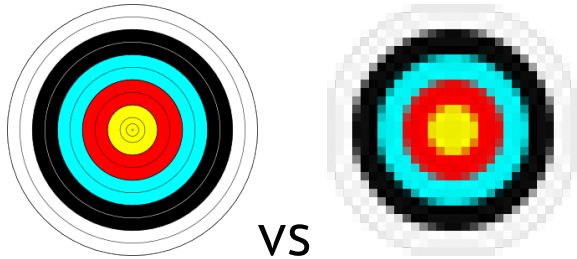
Giving Up?

"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

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

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]
...	

MaxSat Applications

Central to the increasing success:
Advances in MaxSat solver technology

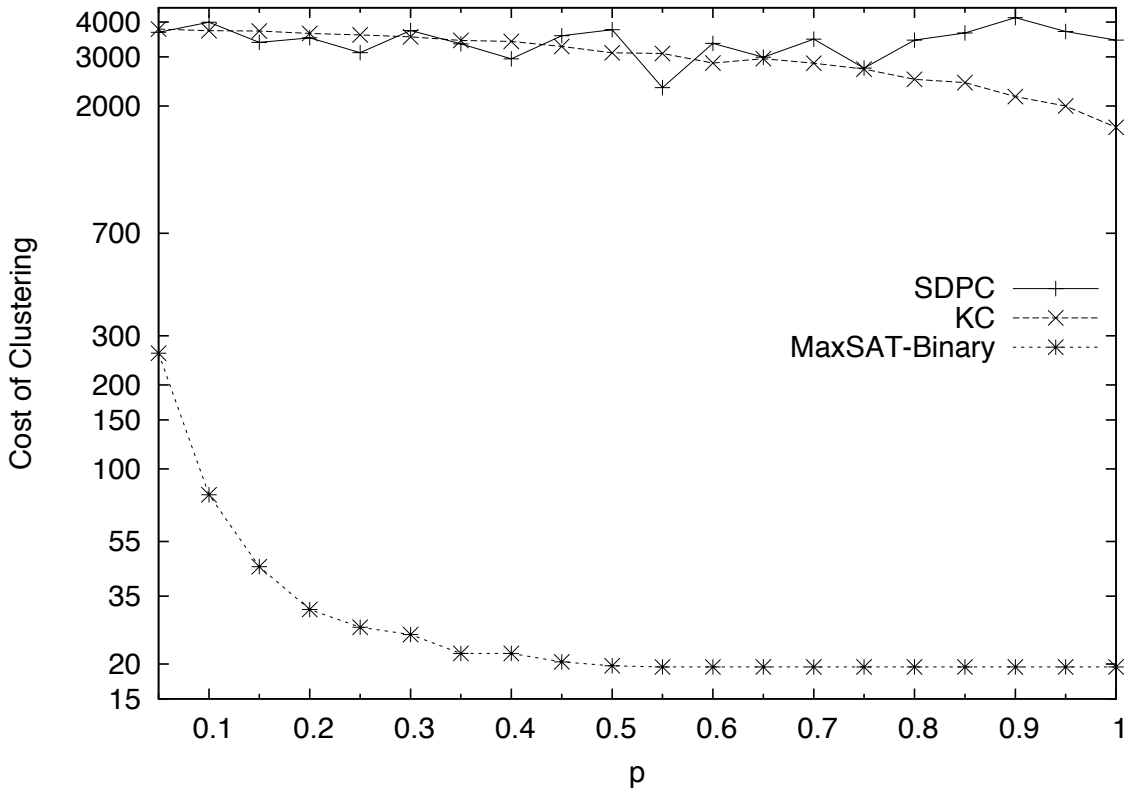
probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]
...	

Benefits of MaxSat

Provably optimal solutions

Example: Correlation clustering by MaxSat

[Berg and Järvisalo, 2016]



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

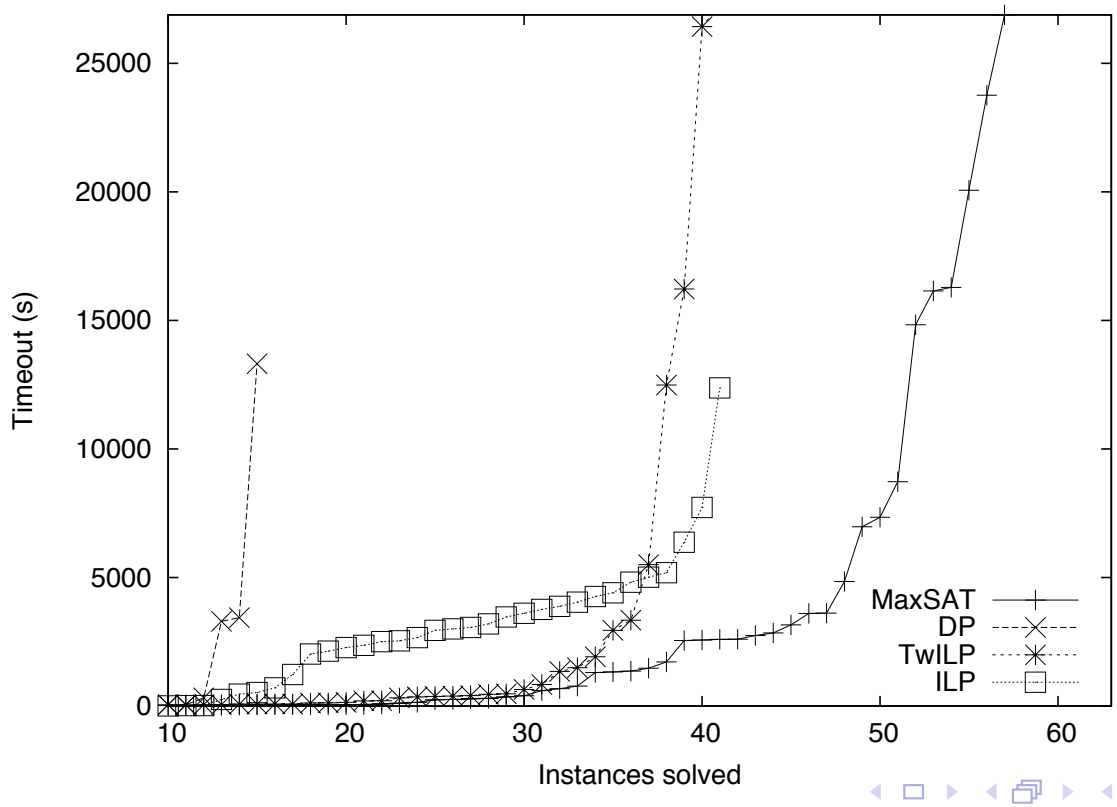
Benefits of MaxSat

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
- 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 τ: a function from Boolean variables to $\{0, 1\}$.
- $\tau(C) = 1$ if
  $\tau(x) = 1$ for a literal $x \in C$,
  $\tau(x) = 0$ for a literal $\neg x \in C$.

  At least one literal of C is made true by τ.

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*

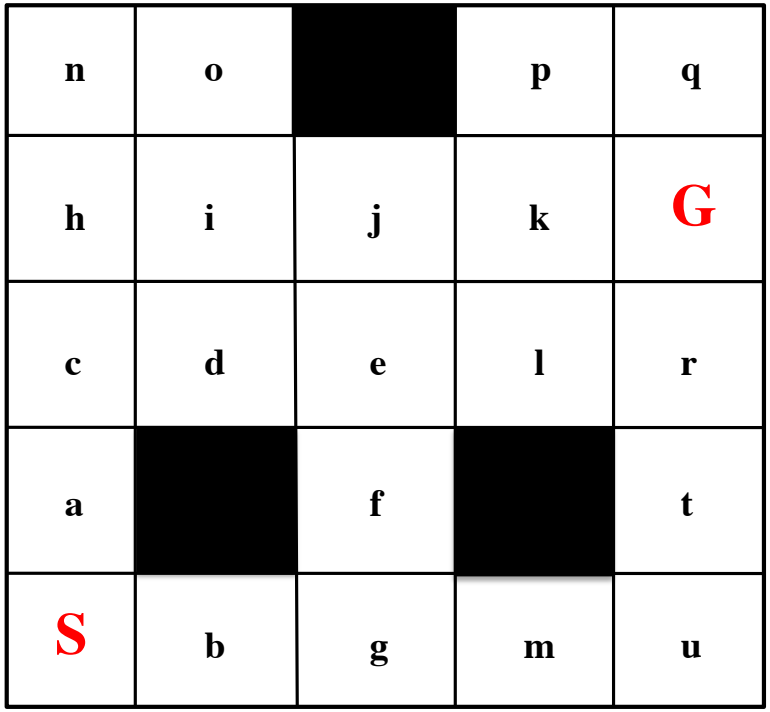
Weighted Partial MaxSat

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

MaxSat: Example

Shortest Path

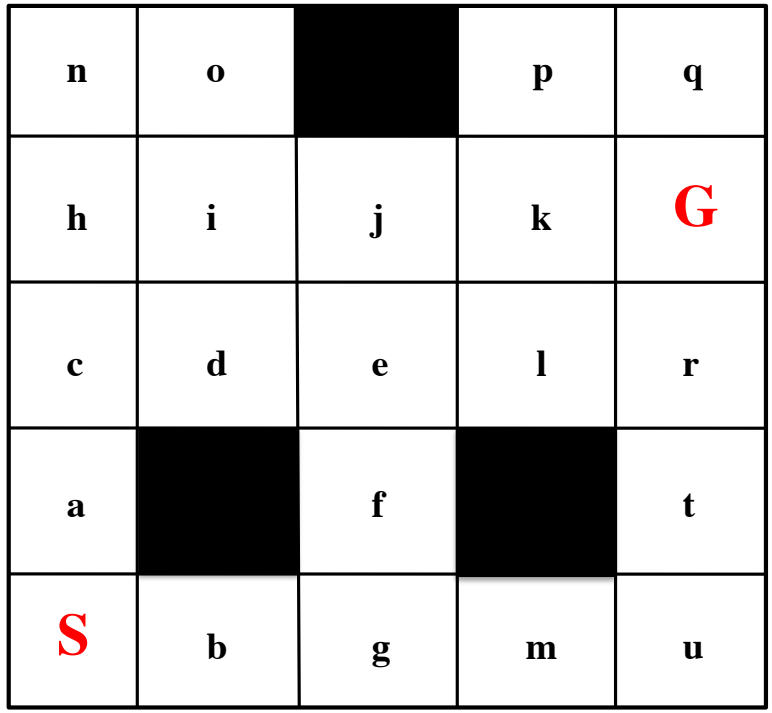
Find shortest path in a grid with horizontal/vertical moves.
Travel from S to G.
Cannot enter blocked squares.



MaxSat: Example

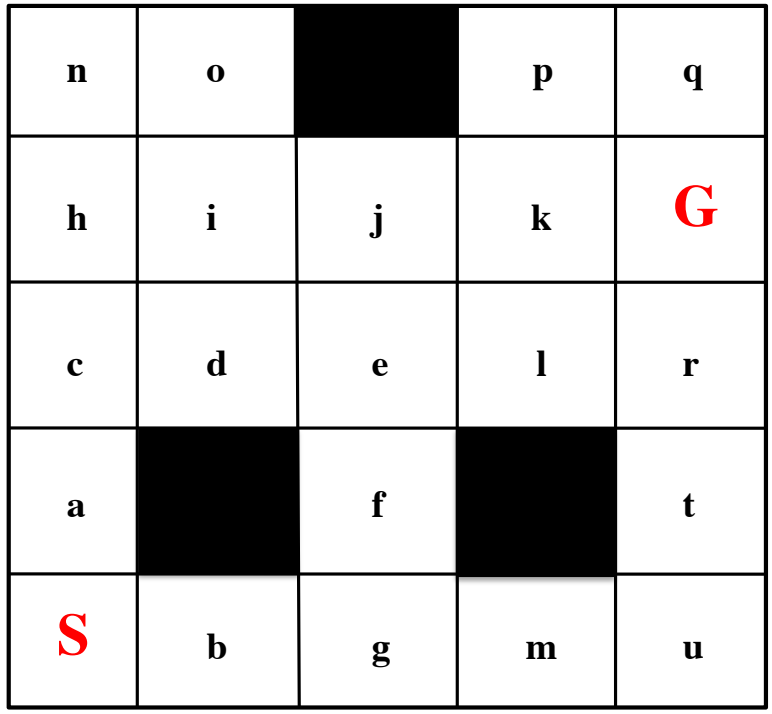
- Note: Best solved with state-space search
  - Used here to illustrate MaxSat encodings

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

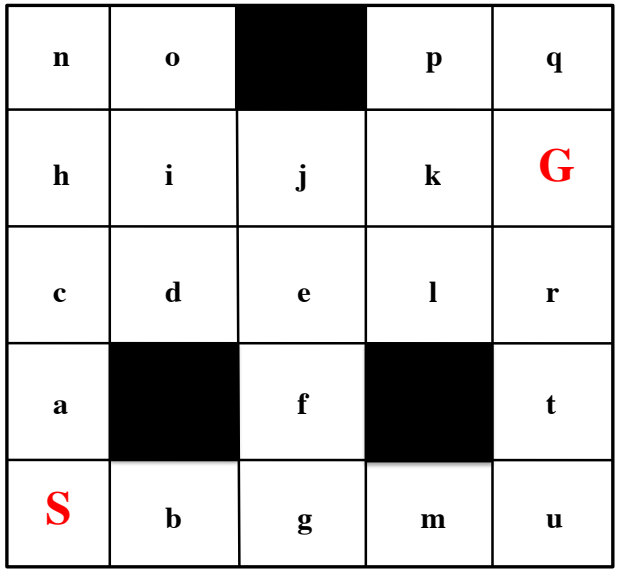
- The previous clauses minimize the number of visited squares.
- ...however, their MaxSat solution will only visit S and G!
- Need to force the existence of a path between S and G by additional hard clauses

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:

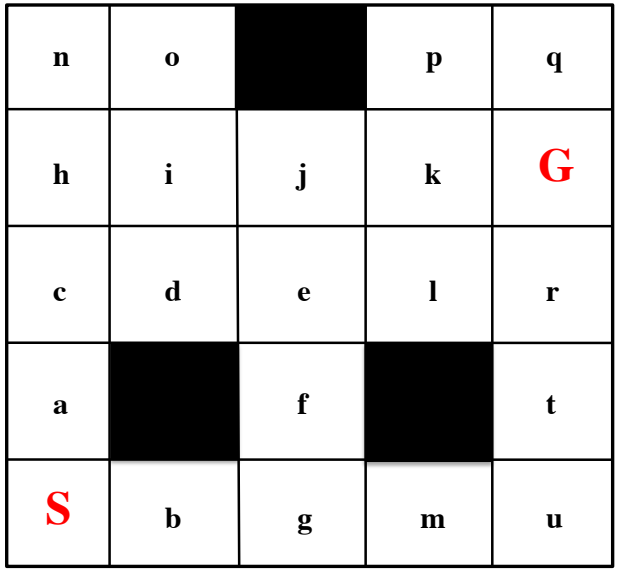
- both S and G must have *exactly one* visited neighbour
  - Any path starts from S
  - Any path ends at G
- other visited squares must have *exactly two* visited neighbours
  - One predecessor and one successor on the path



MaxSat: Example

Constraint 1:

S and G must have exactly one visited neighbour.

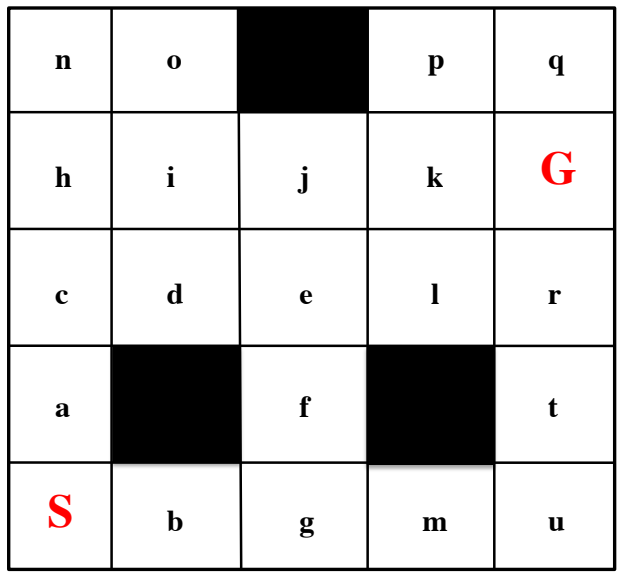


MaxSat: Example

Constraint 1:

S and G must have exactly one visited neighbour.

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



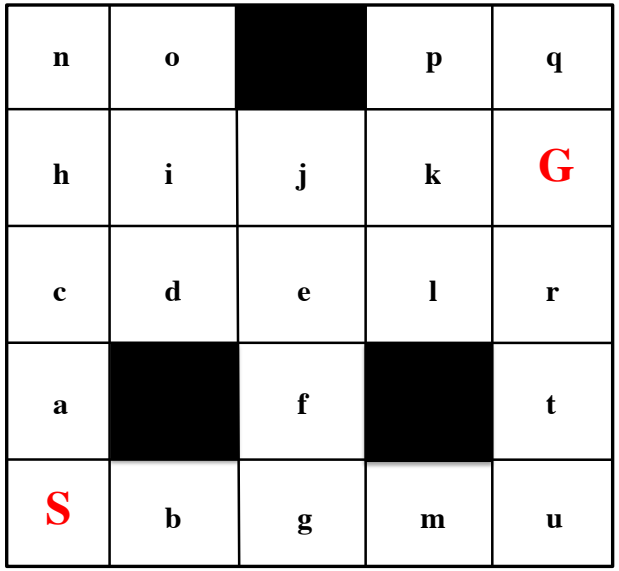
MaxSat: Example

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: $(\neg k \vee \neg q), (\neg k \vee \neg r), (\neg q \vee \neg r)$

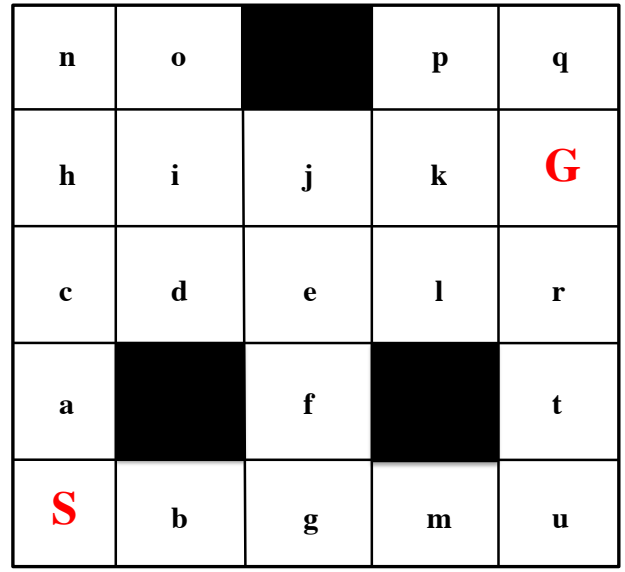
disallow pairwise



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

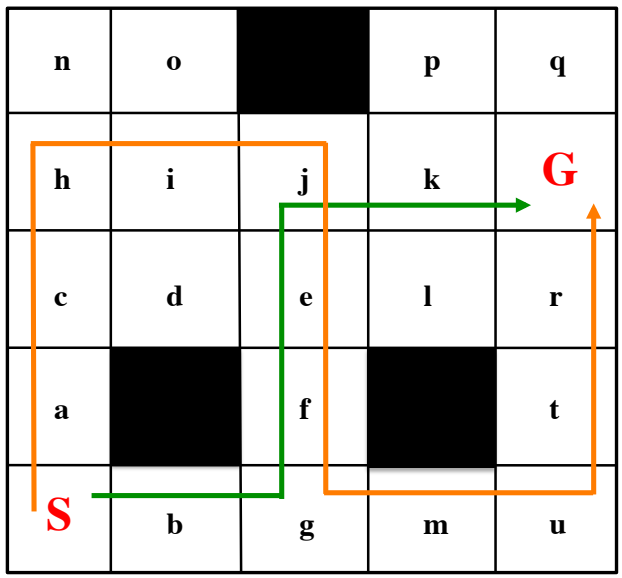
- For example, for square e: $e \rightarrow (d + j + l + f = 2)$
  - Requires encoding the cardinality constraint $d + j + l + f = 2$ in CNF

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

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 $\mathrm{FP}^{\mathrm{NP}}$–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 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).

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

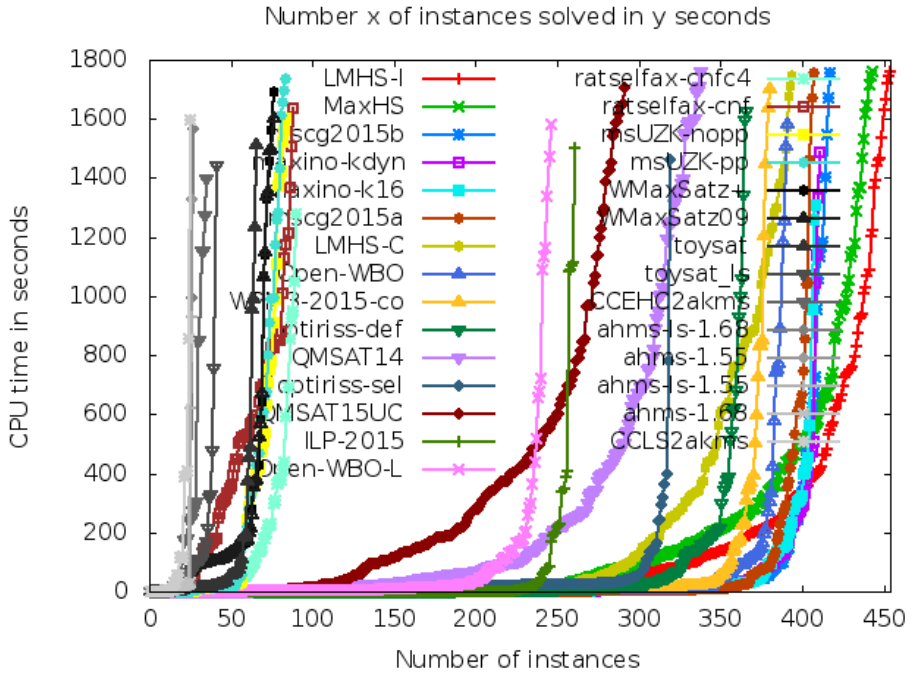
Objectives

- Assessing the state of the art in the field of Max-SAT solvers
- Creating a collection of publicly available Max-SAT benchmark instances
- *Tens of solvers from various research groups internationally* participate each year
- Standard input format

11th MaxSat Evaluation

`http://maxsat.ia.udl.cat`

Affiliated with SAT 2016:
19th Int'l Conference on Theory and Applications of Satisfiability Testing



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

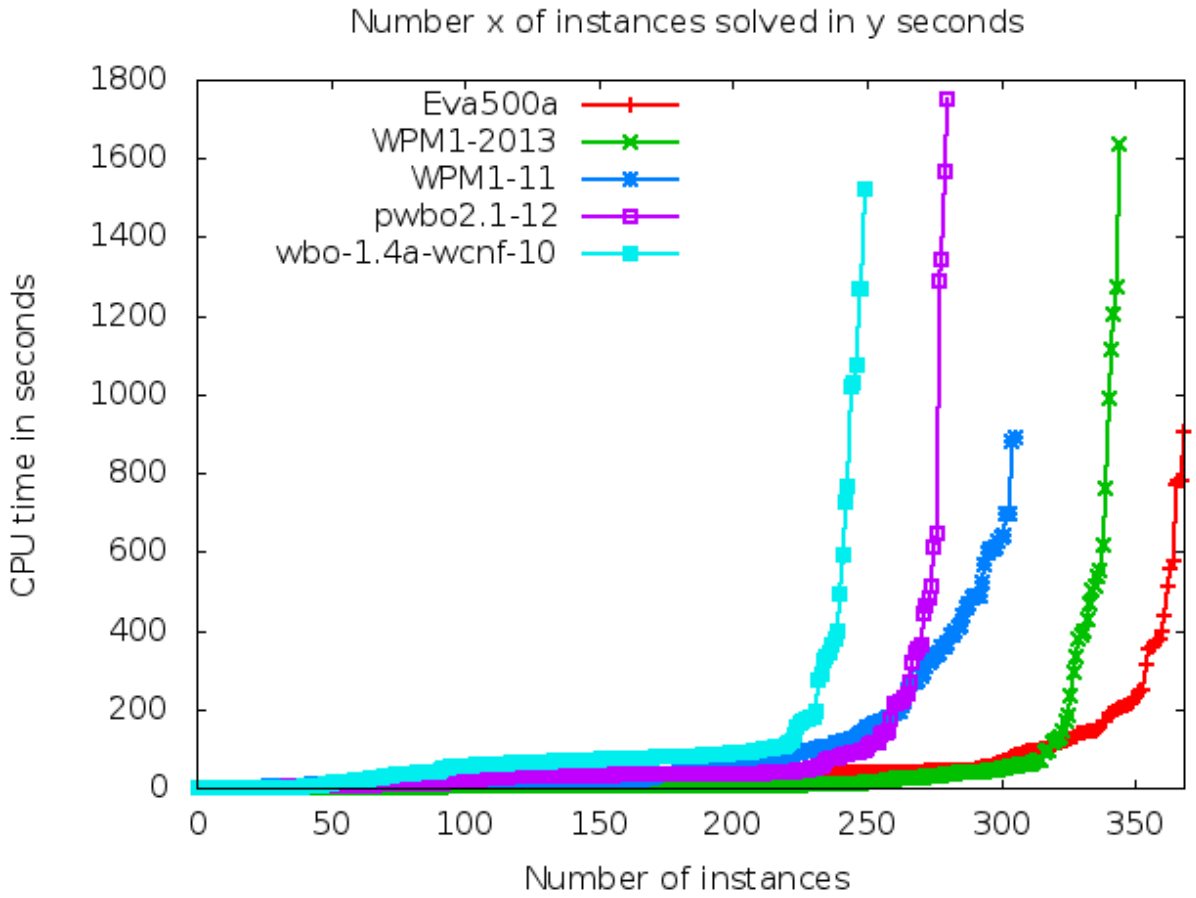
Push-Button Solver Technology

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
```

Progress in MaxSat Solver Performance



Comparing some of the best solvers from 2010–2014:

In 2014: 50% more instances solved than in 2010!

- On same computer, same set of benchmarks: Weighted Partial MaxSat encodings of “industrial”

Some Recent MaxSAT Solvers

Open-source:

- OpenWBO `http://sat.inesc-id.pt/open-wbo/`
- MaxHS `http://maxhs.org`
- LMHS `http://www.cs.helsinki.fi/group/coreo/lmhs/`

Binaries available:

- Eva `http://www.maxsat.udl.cat/14/solvers/eva500a__`
- MaxSatz `http://home.mis.u-picardie.fr/~cli/EnglishPage.html`
- MSCG `http://sat.inesc-id.pt/~aign/soft/`
- WPM3 `http://web.udl.es/usuaris/q4374304/#software`
- QMaxSAT `https://sites.google.com/site/qmaxsat/`
- *...see evaluation web site*
  `http://www.maxsat.udl.cat/15/solvers` *for links to other solvers*

Algorithms for MaxSat Solving

A Variety of Approaches

- Branch-and-bound
- Integer Programming (IP)
- SAT-Based Algorithms
  - Iterative / “model-based”
  - Core-based
- Implicit hitting set algorithms (IP/SAT hybrid).

Examples of Recent MaxSAT Solvers by Category

Branch-and-bound:

- MaxSatz
  http://home.mis.u-picardie.fr/~cli/EnglishPage.html
- ahmaxsat
  http://www.lsis.org/habetd/Djamal_Habet/MaxSAT.html

Iterative, model-based:

- QMaxSAT https://sites.google.com/site/qmaxsat/

Core-based:

- Eva http://www.maxsat.udl.cat/14/solvers/eva500a__
- MSCG http://sat.inesc-id.pt/~aign/soft/
- OpenWBO http://sat.inesc-id.pt/open-wbo/
- WPM http://web.udl.es/usuaris/q4374304/#software
- maxino http://alviano.net/software/maxino/

IP-SAT Hybrids:

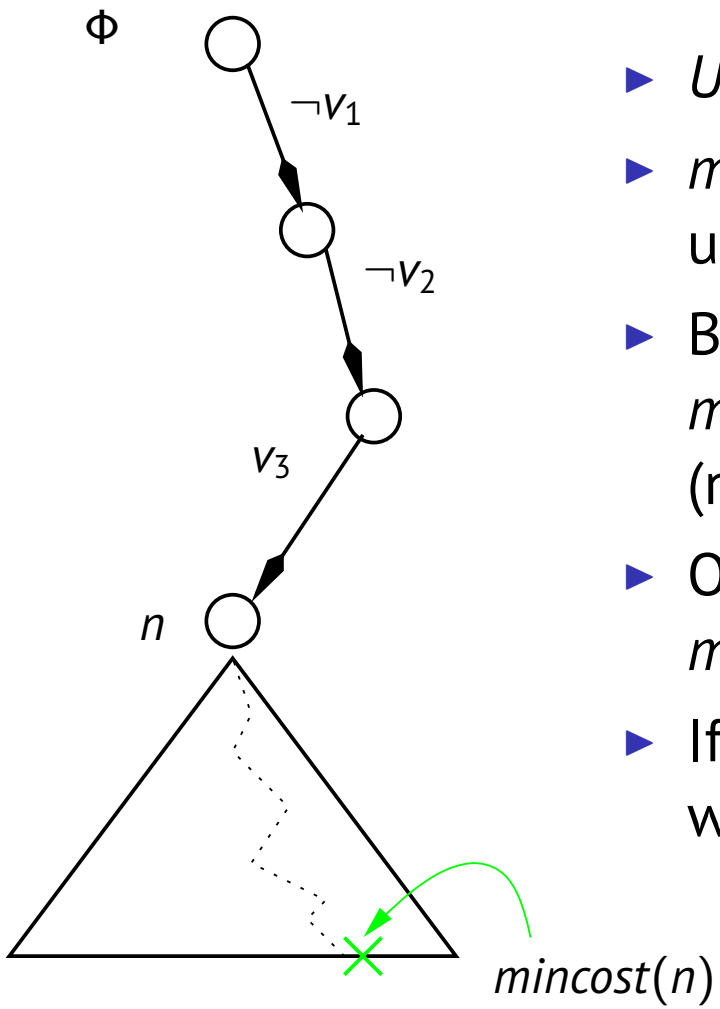
- MaxHS http://maxhs.org
- LMHS http://www.cs.helsinki.fi/group/coreo/lmhs/

Some Additional Notation

- A MaxSat CNF Φ has hard and soft clauses:
  $c_i \in \Phi$: $cost(c_i)$ is weight
  $cost(c_i) < \infty$ soft clause $\quad$ $cost(c_i) = \infty$ hard clause
- Truth assignment π
  $cost(\pi)$ is sum of the weights of clauses falsified by π.
- π satisfies al hard clauses: it is a solution.
- $cost(\pi)$ is minimum over all solutions: it is an optimal solution.
- MaxSat algorithms often cast as **minimization** problem: find solution with minimum *cost* (optimal).

Branch and Bound

Branch and Bound



- $UB =$ cost of the best solution so far.
- $mincost(n) =$ minimum cost achievable under node n
- Backtrack from n when we know $mincost(n) \geq UB$
  (no solution under n is better).
- Our goal: calculate a lower bound LB s.t. $mincost(n) \geq LB$.
- If $LB \geq UB$ then $mincost(n) \geq LB \geq UB$ and we can backtrack.

Lower Bounds

Common LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.

Lower Bounds

Common LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.

$\Phi = \ldots \wedge (x, 2) \ldots \wedge (\neg x, 3) \ldots$
Ignoring clause costs, $\kappa = \{(x) \wedge (\neg x)\}$ is inconsistent.

Lower Bounds

Common LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.

$\Phi = \ldots \wedge (x, 2) \ldots \wedge (\neg x, 3) \ldots$
Ignoring clause costs, $\kappa = \{(x) \wedge (\neg x)\}$ is inconsistent.

Let $\kappa' = \{(\square, 2) \wedge (\neg x, 1)\}$.
Then κ' is MaxSat-equivalent to κ: the cost of each truth assignment is preserved. ($\square$ is empty clause)

Lower Bounds

Common LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.

$\Phi = \ldots \wedge (x, 2) \ldots \wedge (\neg x, 3) \ldots$
Ignoring clause costs, $\kappa = \{(x) \wedge (\neg x)\}$ is inconsistent.

Let $\kappa' = \{(\square, 2) \wedge (\neg x, 1)\}$.
Then κ' is MaxSat-equivalent to κ: the cost of each truth assignment is preserved. ($\square$ is empty clause)

Let $\Phi' = \Phi - \kappa \cup \kappa'$.
Then Φ' is MaxSat-equivalent to Φ, and the cost of $\square$ has been incremented by 2
Cost of $\square$ must be incurred: it is an LB

Lower Bounds

1. Detect an inconsistent subset κ (aka core) of the current formula
   - e.g. $\kappa = \{(x, 2) \wedge (\neg x, 3)\}$
2. Apply sound transformation to the clauses in κ that result in an increment to the cost of the empty clause $\square$
   - e.g. κ replaced by $\kappa' = \{(\square, 2) \wedge (\neg x, 1)\}$
   - This replacement increases cost of $\square$ by 2.
3. Repeat 1 and 2 until no further increment to the *LB* is possible (or $LB \geq UB$)

Fast detection of some Cores

Treat the soft clauses as if they were hard and then:

- Run **Unit Propagation** (UP). If UP falsifies a clause we can find a core.
  On $\{(x, 2), (\neg x, 3)\}$ UP yields false clause.
- The false clause and the clauses that generated it form a core.
- This can find inconsistent sub-formulas quickly
  But only limited set of inconsistent sub-formulas.

Transforming the Formula

- Various sound transformations of cores into increments of the empty clause have been identified.
- **MaxRes** generalizes this to provide a sound and complete inference rule for MaxSat

[Larrosa and Heras, 2005]

[Bonet, Levy, and Manyà, 2007]

MaxRes

- **MaxRes** is a rule of inference that like ordinary resolution takes as input two clauses and produces new clauses.
- Unlike resolution **MaxRes** (a) removes the input clauses and (b) produces multiple new clauses.

MaxRes

MaxRes $[(x \vee a_1 \vee \ldots \vee a_s \vee w_1), (\neg x \vee b_1 \vee \ldots \vee b_t, w_2)] =$

$(a_1 \vee \cdots \vee a_s \cdots b_1 \vee \cdots \vee b_t, \min(w_1, w_2))$	Regular Resolvent
$(x \vee a_1 \vee \cdots \vee a_s, w_1 - \min(w_1, w_2))$	Cost Reduced Input
$(\neg x \vee b_1 \vee \cdots \vee b_t, w_2 - \min(w_1, w_2))$	One will vanish
$(x \vee a_1 \vee \cdots \vee a_s \vee \neg(b_1 \vee \cdots \vee b_t), \min(w_1, w_2))$	Compensation Clauses
$(\neg x \vee \neg(a_1 \vee \cdots \vee a_s) \vee b_1 \vee \cdots \vee b_t, \min(w_1, w_2))$	must be converted to Clauses

[Larrosa and Heras, 2005; Bonet, Levy, and Manyà, 2007]

MaxRes

- By adding the "compensation" clauses **MaxRes** preserves the cost of every truth assignment.
- Bonet et al. give a directly clausal version and a systematic way of using **MaxRes** to derive the empty clause $(\square, \mathit{Opt})$ with weight Opt equal to the optimal cost.

[Bonet, Levy, and Manyà, 2007]

Other Lower Bounding Techniques

- Falsified soft learnt clauses and hitting sets over their proofs

[Davies, Cho, and Bacchus, 2010]

- Clone is an approach that used a relaxation of the MaxSat formula.

[Pipatsrisawat, Palyan, Chavira, Choi, and Darwiche, 2008]

- The relaxation provides a LB at each node.
- Other relaxations including minibuckets, or width-restricted BDDs might be applied.

[Dechter and Rish, 2003]

[Bergman, Ciré, van Hoeve, and Yunes, 2014]

Branch and Bound Summary

- Can be effective on small combinatorially hard problems, e.g., maxclique in a graph.
- Once the number of variables gets to 1,000 or more it is less effective: LB techniques become weak or too expensive.

MaxSat by Integer Programming (IP)

Solving MaxSat with an IP Solver

- Optimization problems studied for decades in operations research (OR).
- Integer Program (IP) Solvers are common optimization tool for OR. E.g., IBM's CPLEX.
- IP solvers solve problems with linear constraints and objective function where some variables are integers.
- State-of-the-art IP solvers very powerful and effective: can use this tool for MaxSat as well.

Blocking Variables (Relaxation Variables)

MaxSat solving uses technique of blocking variables to relax (block) soft clauses (selector variables).

- To a soft clause $(x_1 \vee x_2 \vee \cdots \vee x_k)$ we add a **new** variable b:

$$(b \vee x_1 \vee x_2 \vee \cdots \vee x_k)$$

b does not appear anywhere else in the formula.

- If we make b true the soft clause is automatically satisfied (is relaxed/is blocked).
- If we make b false the clause becomes hard and must be satisfied.

MaxSat encoding into IP

- To every soft clause c_i add a new "blocking" variable b_i.

$$(x \vee \neg y \vee z \vee \neg w) \Rightarrow (b_1 \vee x \vee \neg y \vee z \vee \neg w)$$

- Convert every augmented clause into a linear constraint:

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

- Each variable is integer in the range $[0 - 1]$.
- Finally add the objective function

$$\text{minimize} \sum_i b_i * cost(c_i)$$

Integer Programming Summary

- IP solvers use Branch and Cut to solve.
  - Compute a series of linear relaxations and cuts (new linear constraints that cut off non-integral solutions).
  - Sometimes branch on a bound for an integer variable.
  - Also use many other techniques.
- Effective on many standard optimization problems, e.g., vertex cover.
- But for problems where there are many boolean constraints IP is not as effective.

SAT-Based MaxSat Solving

SAT-Based MaxSat Solving

- Solve a sequence or SAT instances where each instance **encodes** a *decision problem* of the form

  "*Is there a truth assignment of falsifying at most weight k soft clauses?*"

  for different values of k.

- SAT based MaxSat algorithms mainly do two things
  1. Develop better ways to encode this decision problem.
  2. Find ways to exploit information obtained from the SAT solver at each stage in the next stage.

Assume unit weight soft clauses for now

SAT-Based MaxSat: Basic Framework

Basic Framework (UNSAT $\Rightarrow$ SAT). We successively relax the MaxSat formula allowing more and more soft clauses to be falsified.

1. Verify that the set of hard clauses are SAT
   If UNSAT STOP. There are no MaxSat solutions!
2. Else: Repeat until Φ is SAT
   - 2.1 Try to SAT solve Φ.
   - 2.2 If SAT STOP found optimal solution
   - 2.3 Else relax Φ so that more soft clauses can be falsified.
     - Minimum relaxation $\Rightarrow$ optimal solution when Φ is SAT.

SAT-Based MaxSat Solving

- Iterative Search methods
- Improving by using Cores
- Improving by using Cores and new variables

Iterative SAT solving
Linear Search

Simplest (and least effective) linear search approach. (Unit clause weights).

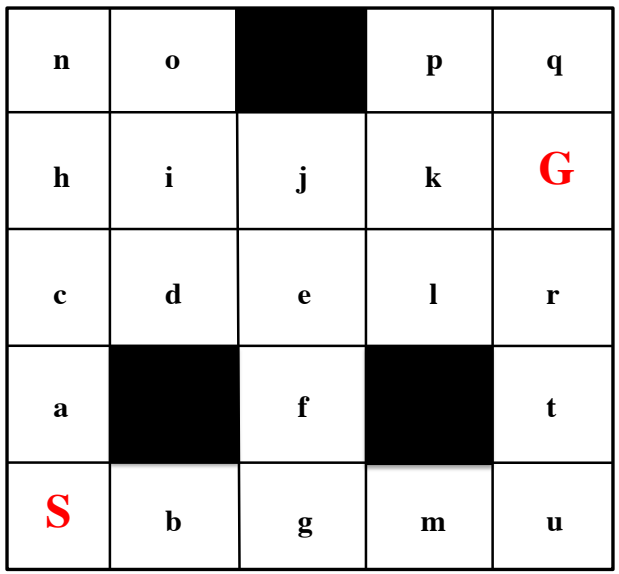
1. Input MaxSat CNF Φ
2. Add blocking variable b_i to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If SAT($\Phi \cup CNF(\sum b_i \leq k)$) return k
5. Else $k = k + 1$ and repeat 4.

Iterative SAT solving (Linear Search)

1. Input MaxSat CNF Φ
2. Add blocking variable b_i to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If $\text{SAT}(\Phi \cup \mathit{CNF}(\sum b_i \leq k))$ return k
5. Else $k = k + 1$ and repeat 4.

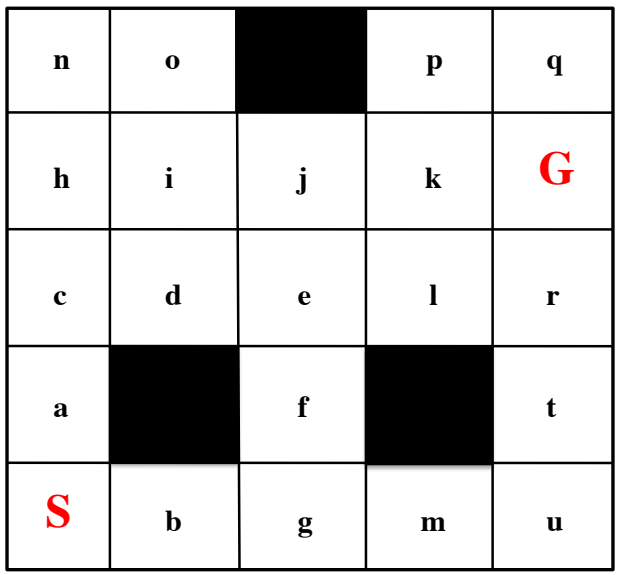
- *CNF* converts cardinality constraint to CNF. By allowing k of the b_i's to be true we "remove" k soft clauses.
- SAT: Try to satisfy remaining clauses after removing any set of up to k soft clauses (the SAT solver searches for which ones to remove).
- If k yields UNSAT we try removing $k + 1$ soft clauses.
- If k yields SAT, prior UNSAT for $k - 1$ proves that the satisfying assignment is optimal.

Iterative SAT solving (Linear Search)



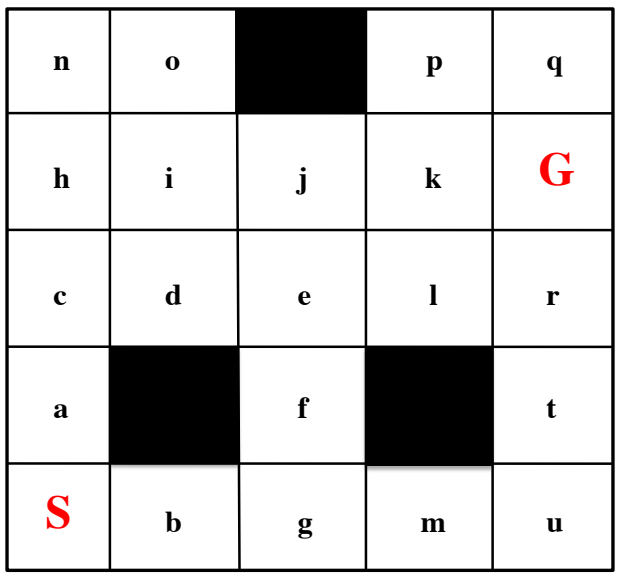
1. Add blocking variables to soft clauses: $(\neg a), \ldots, (\neg t) \mapsto (\neg a \vee b_a), \ldots, (\neg u \vee b_u)$.
2. When $k = 0$ cardinality constraint $\sum b_i \leq k$ forces $\neg b_a, \neg b_b, \ldots, \neg b_u$.
3. This in turn forces $\neg a, \neg b, \ldots, \neg u$.
4. Hard clause $(a \vee b)$ (must exit S) falsified $\Rightarrow$ UNSAT.
   - Hard clause $(k \vee q \vee r)$ (must enter G) also falsified.
5. Increment k to $k = 1$.

Iterative SAT solving (Linear Search)



1. Increment $k = 1$.
2. If the solver tries to set a square like e to true:
   - the clause $(\neg e \vee b_e)$ forces b_e.
   - the cardinality constraint forces all other b's to be false and these force all squares to false.
   - Hard constraint $e \rightarrow (d + j + l + f = 2)$ is falsified and the solver forces $\neg e$.
3. If the solver sets a (neighbour of S) to true
   - force b_a, and b-variables and squares to be false.
   - $a \rightarrow (S + c = 2)$ is falsified.
   - solver forces $\neg a$. And from $(a \vee b)$ forces b.
   - $b \rightarrow (S + g = 2)$ is falsified.
   - UNSAT.

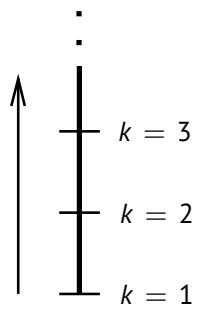
Iterative SAT solving (Linear Search)



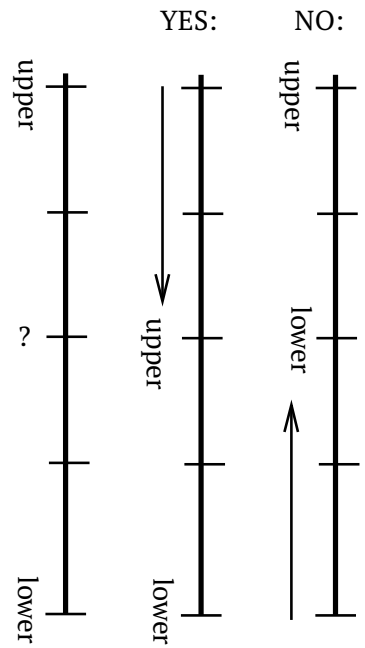
- SAT solver will examine longer paths as k gets larger.
- A path must exit S. Path can visit at most k squares. Must end at G else obtain conflict: the last square will have not have two visited neighbours.
- When all paths of length k from S are refuted we get UNSAT.
- Only when k is large enough to admit a path from S to G will we get SAT.
- The smallest value of k will be found, and the satisfying assignment will specify a shortest path.

Iterating over k

- Different ways of iterating over values of k.
- Three "standard" approaches:



1. Linear search (not effective)
   - Start from $k = 1$.
   - Increment k by 1 until a solution is found.
2. Binary search (used effectively in MSCG when core based reasoning is added)
   - $UB =$ # of soft clauses; $LB = 0$.
   - Solve with $k = (UB + LB)/2$.
   - if SAT: $UB = k$; if UNSAT: $LB = k$
   - When $UB = LB + 1$, UB is solution.

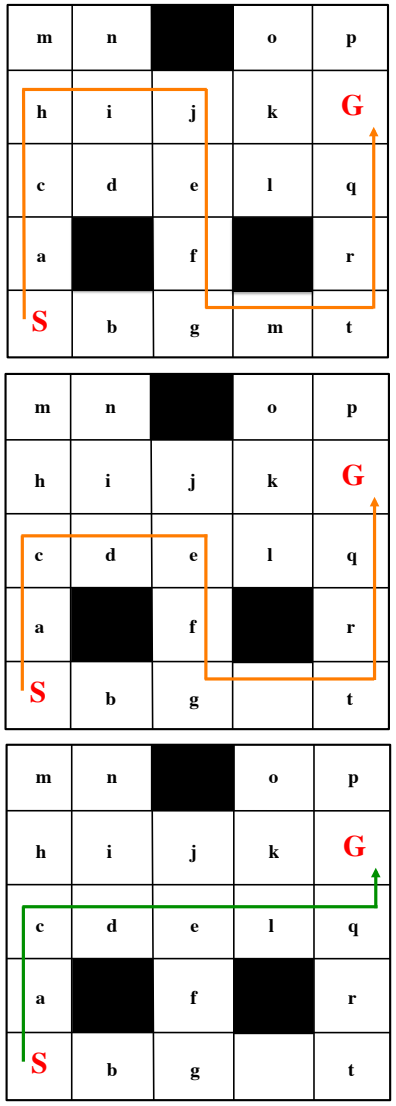


Iterating over k

3. SAT to UNSAT (used in QMaxSAT, can be effective on certain problems)
   - 3.1 Find a satisfying assignment π of the hard clauses.
   - 3.2 Solve with $k =$ (# of clauses falsified by π) $- 1$
   - 3.3 If SAT found better assignment. Reset k and repeat 2.
   - 3.4 If UNSAT last assignment π found is optimal.

This method finds a sequence of improved models—thus can give an approximate solution.

Iterative SAT to UNSAT



- The SAT solver must find a shorter path at each stage.

SAT-based MaxSat Solving using Cores

SAT-Based MaxSat using Cores

Core

Given an unsatisfiable CNF formula Φ, a **core** of Φ is a subset of Φ that is itself unsatisfiable.

Cores for MaxSat

A subset of soft clauses of Φ that together with the **hard clauses** of Φ is unsatisfiable.

Cores from SAT Solvers

- Modern SAT solvers can return a core when input is UNSAT.
- By removing the hard clauses from the core, we obtain a core for MaxSat
- $(\kappa, \mathit{SAT}?) = \mathsf{sat}(\Phi)$
  Sat solve Φ. Return Boolean SAT or UNSAT status: *SAT*?
  If UNSAT, return a core κ (set of soft clauses).
- Different methods are available for obtaining the core:
  1. Using assumptions.
  2. Outputting a clausal proof and then obtaining a core from trimming it.

Core-Based MaxSat Solving

Improvement over iterative methods

- In the linear approach we add $CNF(\sum b_i \leq k)$ to the SAT solver.
- There is one b_i for every soft clause in the theory. This cardinality constraint could be over 100,000s of variables: it is very loose.
  No information about which particular blocking variables to make true.
- This makes SAT solving inefficient: could have to explore many choices of subsets of k soft clauses to remove.
- However, if we obtain a core we have a powerful constraint on which particular soft clauses need to be removed.

Constraint from Cores

- If κ is a MaxSat core, then at least one if the soft clauses in it must be removed: no truth assignment satisfies every clause in κ along with all of the hard clauses.
- Typically cores are *much* smaller than the set of all soft clauses.

MSU3

MSU3 is an simple MaxSat algorithm for exploiting cores

[Marques-Silva and Planes, 2007].

- Only adding blocking variables to soft clauses that appear in a core.
- $CNF(\sum b_i \leq k)$ generally remains over much smaller set of variables.

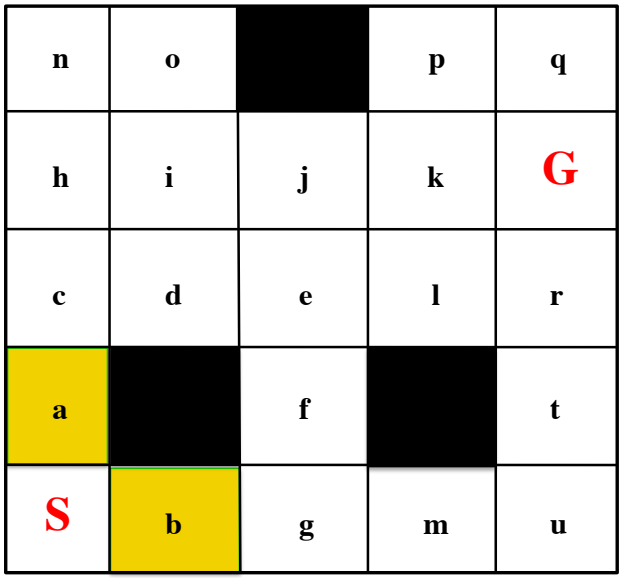
  cardinality constraint much tighter

MSU3

1. Input MaxSat CNF Φ
2. $k = 0$; $BV = \{\}$.
3. $(\kappa, SAT?) = \text{sat}(\Phi)$
4. If $SAT?$ **return** k.
5. $k = k + 1$
6. Update Φ:
   - 6.a For $c \in \kappa$ if c has no blocking variable
     $c = c \cup \{b\}$ (new blocking variable)
     $BV = BV \cup \{b\}$
   - 6.b Remove previous cardinality constraint.
   - 6.c Add $CNF(\sum_{b \in BV} b \leq k + 1)$
7. GOTO 3

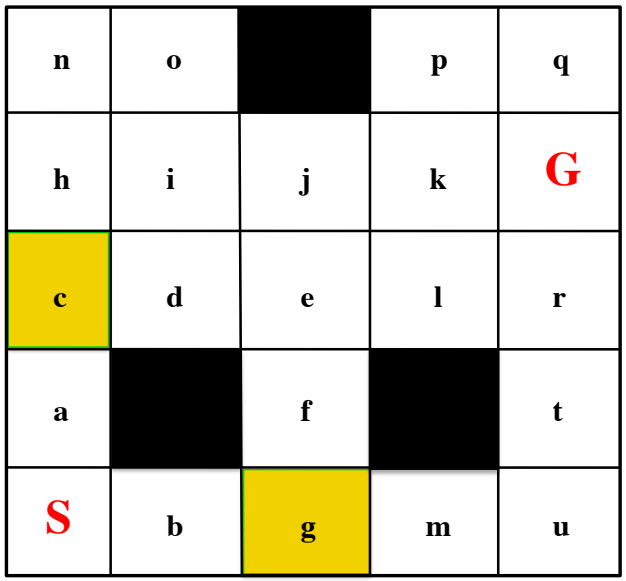
- Initially NO blocking variables!
- The cardinality constraint is always only over soft clauses that have participated in some core.
- The blocking variables in the cardinality constraint grows as more cores are discovered.
- On many problems however the cardinality constraint remains over a proper subset of the soft clauses.

MSU3



1. $\kappa = \{(\neg a)(\neg b)\}$ is one possible core when $k = 0$.
2. Update these soft clauses to add a blocking variable to each: $\{(\neg a, b_a), (\neg b, b_b)\}$.
3. Add $CNF(b_a + b_b \leq 1)$
4. $k = 1$
5. SAT solve again.

MSU3



1. $\kappa = \{(\neg c)(\neg g)\}$ is a possible core in the updated formula.
2. None has a blocking variable as yet—add: $\{(\neg c, b_c), (\neg g, b_g)\}$.
3. Remove previous cardinality constraint.
4. Add $CNF(b_a + b_b + b_c + b_g \leq 2)$
5. $k = 2$
6. SAT solve again.

MSU3

1. By itself MSU3 is not effective.
2. Very effective when combined with an incremental construction of the cardinality constraint (so that each new constraint builds on the encoding of the previous constraint).

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

3. OpenWBO uses MSU3 with incremental cardinality constraints to achieve state-of-the-art performance on many problems.

Stronger Core Constraints

- In 2nd iteration MSU3 used the cardinality constraint $(b_a + b_b + b_c + b_g \leq 2)$.
- At this stage actually know something stronger: $(b_a + b_b \leq 1)$ and $(b_c + b_g \leq 1)$.
- In the Fu-Malik algorithm each core found is encoded as a separate cardinality constraint.

[Fu and Malik, 2006]

- So Fu-Malik would at the 2nd iteration use the stronger constraint $(b_a + b_b \leq 1) \wedge (b_c + b_g \leq 1)$.

Overlapping Cores

- However, overlapping cores pose a problem!
- Say the first and second cores are
  1. $\{(\neg a), (\neg b)\}$
  2. $\{(\neg b, b_b), (\neg c), (\neg g)\}$
- The soft clause $(\neg b)$ participates in both cores!
- Core 2 is a core of the updated formula that includes $(\neg a \vee b_a), (\neg b \vee b_b)$ and $(b_a + b_b \leq 1)$: one of a or b can be true.
- Core 2 is an unsatisfiable set of soft clauses even when we are allowed to set one of a or b to true.
  Core 2 and Core 1 imply that we have

$$\big(a \wedge (b \vee c \vee g)\big) \vee \big(b \wedge (c \vee g)\big)$$

Overlapping Cores

- For these cores the constraint $(b_a + b_b \leq 1) \wedge (b_b + b_c + b_g \leq 1)$ is too strong: E.g. $b \wedge c$ is a solution of

$$\big(a \wedge (b \vee c \vee g)\big) \vee \big(b \wedge (c \vee g)\big)$$

but is not a solution of the two cardinality constraints

$$(b_a + b_b \leq 1) \wedge (b_b + b_c + b_g \leq 1)$$

- Dealing with overlapping cores is a complicating issue for most core-based algorithms.

Fu-Malik

Fu-Malik deals with overlapping cores by adding a new blocking variables to the clauses in the core even when they already have a previous blocking variable.

1. $k = 0, \kappa = \{(\neg a), (\neg b)\}$
2. Update these clauses to $\{(\neg a, b_a), (\neg b, b_b)\}$.
3. Add $CNF(b_a + b_a \leq 1)$
4. $k = 1, \kappa = \{(\neg b, b_b), (\neg c), (\neg g)\}$
5. Update these clauses to $\{(\neg b, b_b, b_b^1), (\neg c, b_c), (\neg g, b_g)\}$
6. Add $CNF(b_b^1 + b_c + b_g \leq 1)$

Fu-Malik

1. Multiple blocking variables in the same soft clause are redundant—lead to symmetric assignments that must be refuted by the SAT solver.

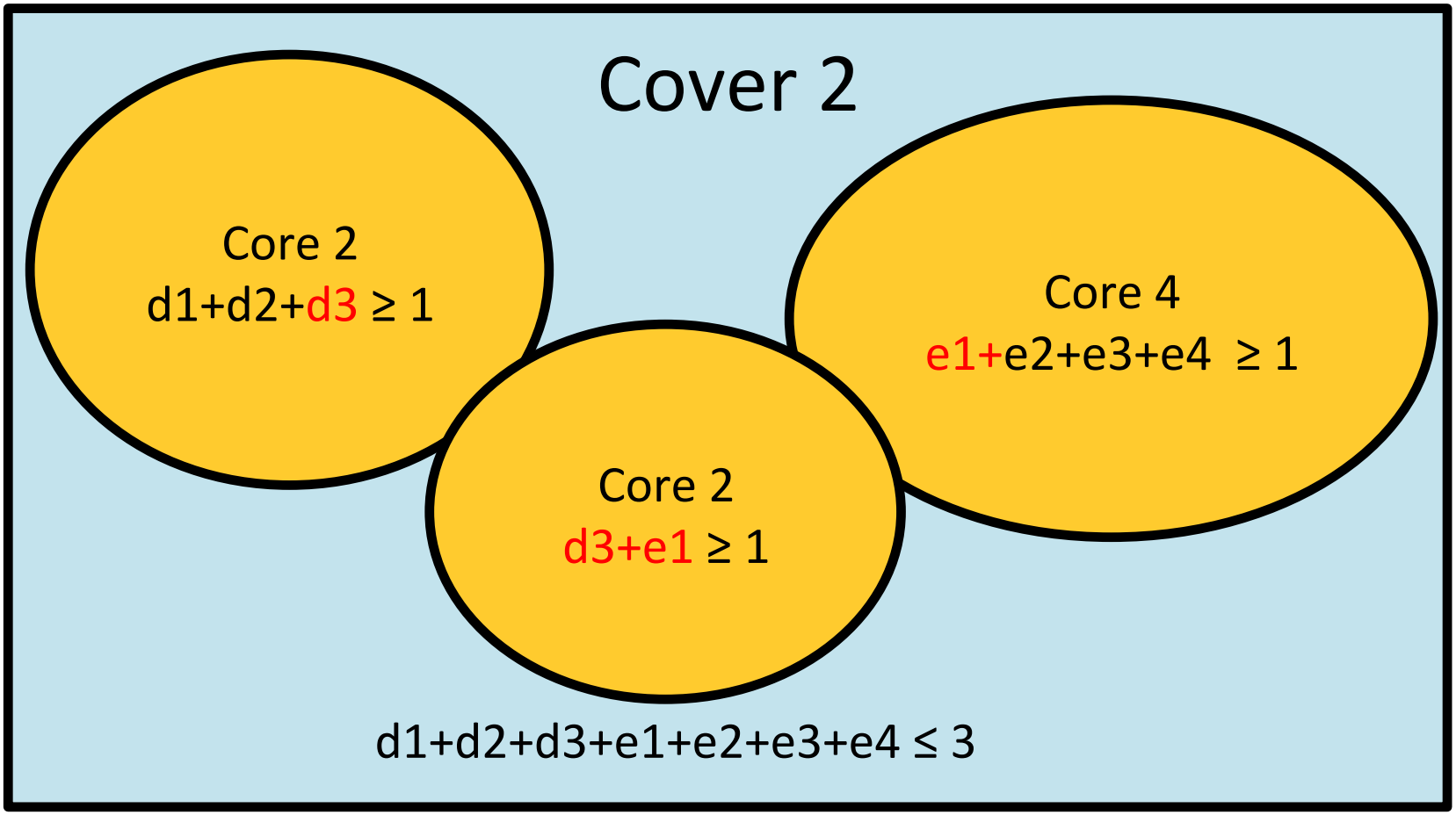
   [Ansótegui, Bonet, and Levy, 2013a]

2. Cardinality constraint is always ≤ 1 so can be encoded more efficiently.
3. Fu-Malik contains some key ideas but no longer state of the art.

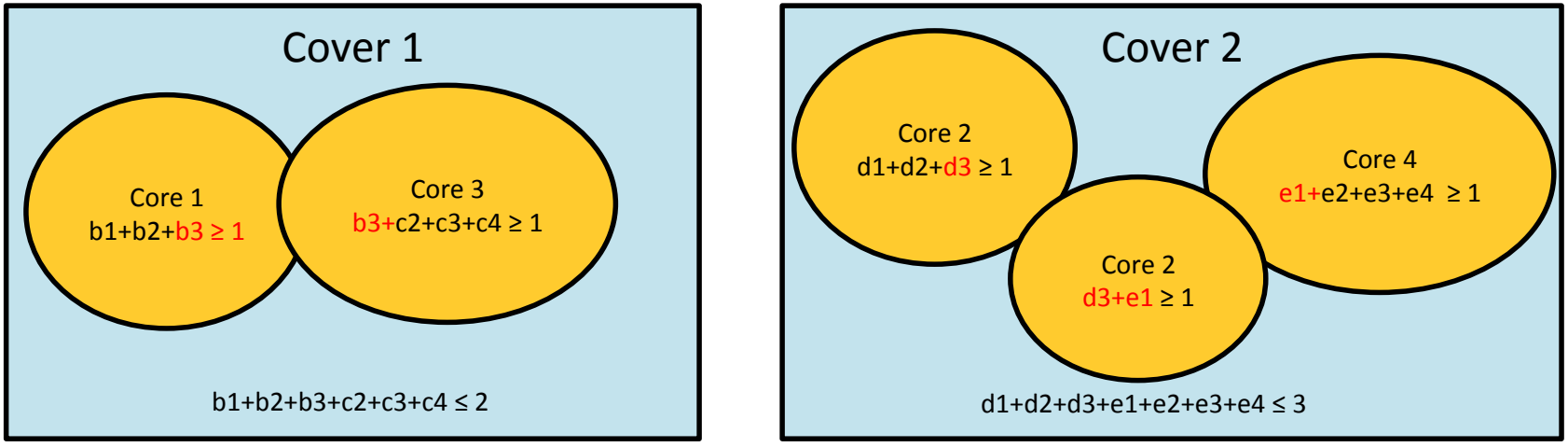
WPM2

- Another method for dealing with overlapping cores was developed by Ansótegui et al. [Ansótegui, Bonet, and Levy, 2013a]
- Only one blocking variable per soft clause.
- Group intersecting cores into disjoint covers.
  The cores might not be disjoint but the covers will be.
- Put a distinct at-most $\leq$ cardinality constraint over the soft clauses in a cover
  Disjoint so this works.
- Keep an at-least $\geq$ constraint over the clauses in a core.

WPM2


Cover 2
Core 2
d1+d2+d3 ≥ 1
Core 2
d3+e1 ≥ 1
Core 4
e1+e2+e3+e4 ≥ 1
d1+d2+d3+e1+e2+e3+e4 ≤ 3

WPM2



- Cover 1 and Cover 2 are disjoint sets of soft clauses.
- Each core in a cover has a non-empty intersection with another core in the cover (only place cores in the same cover if you have to)

WPM2

- When each new core is found covers must be adjusted.
- Each core in the cover found one after the other.
- A soft clause of each core must be blocked even though we have already blocked a clause from all prior cores.
- Cover at-most bound equal to sum of its core at-least bounds.

WPM2

- Each new core generates an update to the set of covers
- Core might make covers non-disjoint: these have to be unioned into one cover.
- Cover at-least must be updated.
- At least one at-least constraint is relaxed—so eventually formula must become SAT.

Cores plus New Variables

State-of-the-art Core based MaxSat

- Recent advances in SAT-Based MaxSat solving comes from approaches that add new variables to the formula.
- New variables always been used encoding the cardinality constraint

  but no attention was paid to the structure of these variables.
- Current best SAT-Based approaches EVA, MSCG-OLL, OpenWBO, WPM3, MAXINO use cores and add new variables.
- EVA, MSCG-OLL and WPM3 explicitly add new variables.
- OpenWBO and MAXINO more carefully structure the new variables in the cardinality constraints.

Eva

Processing a new core in EVA [Narodytska and Bacchus, 2014]

1. Core $= \{c_1, c_2, c_3, c_4, c_5\}$ New core

Eva

Processing a new core in EVA [Narodytska and Bacchus, 2014]

1. Core $= \{c_1, c_2, c_3, c_4, c_5\}$ New core
2. $b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3$ new variables b_i equivalent to c_i
   $b_4 \equiv c_4 \quad b_5 \equiv c_5$

Eva

Processing a new core in EVA [Narodytska and Bacchus, 2014]

1. Core $= \{c_1, c_2, c_3, c_4, c_5\}$ — New core
2. $b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3$
   $b_4 \equiv c_4 \quad b_5 \equiv c_5$ — new variables b_i equivalent to c_i
3. Remove softs $\{c_1, c_2, c_3, c_4, c_5\}$

Eva

Processing a new core in EVA [Narodytska and Bacchus, 2014]

1. Core $= \{c_1, c_2, c_3, c_4, c_5\}$ — New core
2. $b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3$
   $b_4 \equiv c_4 \quad b_5 \equiv c_5$ — new variables b_i equivalent to c_i
3. Remove softs $\{c_1, c_2, c_3, c_4, c_5\}$
4. $(\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee \neg b_4 \vee \neg b_5)$ — must falsify one of the c_i

Eva

Processing a new core in EVA

[Narodytska and Bacchus, 2014]

1. Core $= \{c_1, c_2, c_3, c_4, c_5\}$ — New core
2. $b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3$
   $b_4 \equiv c_4 \quad b_5 \equiv c_5$ — new variables b_i equivalent to c_i
3. Remove softs $\{c_1, c_2, c_3, c_4, c_5\}$
4. $(\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee \neg b_4 \vee \neg b_5)$ — must falsify one of the c_i
5. $d_1 \equiv b_2 \wedge b_3 \wedge b_4 \wedge b_5$
   $d_2 \equiv b_3 \wedge b_4 \wedge b_5$
   $d_3 \equiv b_4 \wedge b_5$
   $d_4 \equiv b_5$
   — new variables d_i; d_i indicates $c_{i+1} \ldots c_5$ satisfied

Eva

Processing a new core in EVA

[Narodytska and Bacchus, 2014]

1. $\text{Core} = \{c_1, c_2, c_3, c_4, c_5\}$ — New core
2. $b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3$
   $b_4 \equiv c_4 \quad b_5 \equiv c_5$ — new variables b_i equivalent to c_i
3. Remove softs $\{c_1, c_2, c_3, c_4, c_5\}$
4. $(\neg b_1 \vee \neg b_2 \vee \neg b_3 \vee \neg b_4 \vee \neg b_5)$ — must falsify one of the c_i
5. $d_1 \equiv b_2 \wedge b_3 \wedge b_4 \wedge b_5$
   $d_2 \equiv \quad b_3 \wedge b_4 \wedge b_5$
   $d_3 \equiv \quad b_4 \wedge b_5$
   $d_4 \equiv \quad b_5$ — new variables d_i; d_i indicates $c_{i+1} \ldots c_5$ satisfied
6. $(b_1 \vee d_1, 1)$
   $(b_2 \vee d_2, 1)$
   $(b_3 \vee d_3, 1)$
   $(b_4 \vee d_4, 1)$ — new softs

Eva

New softs relax the formula, e.g., falsify one soft.

- Falsify one soft clauses from $\{c_1, \ldots, c_5\}$ (say c_3):
  1. $b_1, b_2, \neg b_3, b_4, b_5$.
  2. Most new softs $(b_i \vee d_i, 1)$ satisfied by b_i.
  3. Consider $(b_3, d_3, 1)$
     $d_3 \equiv b_4 \wedge b_5$ is TRUE
     so this soft clause is satisfied.

  No cost is incurred in the new formula. (new formula is relaxed)

Eva

New softs relax the formula, e.g., falsify more than one soft.

- Falsify two soft clauses (say c_2 and c_3):
  1. $b_1, \neg b_2, \neg b_3, b_4, b_5$.
  2. $(b_1 \vee d_1, 1), (b_4 \vee d_4, 1)$ satisfied.
  3. $d_2 \equiv b_3 \wedge b_4 \wedge b_5$ is FALSE
     $(b_2 \vee d_2, 1)$ is FALSIFIED.
  4. $d_3 \equiv b_4 \wedge b_5$ is TRUE
     $(b_3 \vee d_3, 1)$ is satisfied

  So if 2 of the c_i are falsified only one new soft clause is falsified.

Eva

- d_i variables capture a disjunction of soft clauses (c.f. extended resolution)
- If a future core involves $(\neg b_i \vee \neg d_i, 1)$ we get a new variable $x_i \equiv (\neg b_1 \vee \neg d_i)$.
- So new variables can build up to represent complex conditions.
- These variable seem to help the SAT solver in finding new cores.
- But a deeper understanding of this has not yet been developed

Soft Cardinality Constraints

- MSCG-OLL and WPM3 introduce new variables to represent the cardinality constraints.
- $d \equiv b_1 + b_2 + b_3 + b_4 + b_5 \leq 1$.
- Soft clause $(d, 1)$ is introduced.
- $(d, 1)$ is falsified if $b_1 + b_2 + b_3 + b_4 + b_5 > 1$.
- The $(d, 1)$ soft clauses can participate in new cores.
- Again these variables seem to help the SAT solver in finding new cores.

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

[Ansótegui, Didier, and Gabàs, 2015]

New Variables in Cardinality Constraints

- openWBO and MAXINO develop special methods for constructing the cardinality constraints associated with each core.
- They build them in such a way that each new cardinality constraint can share variables with the previous constraints.
- This tends to generate new variables expressing the sum of useful sets of soft constraints (soft constraints that appear together in more than one core).
- Again these variable seem to help the SAT solver

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

[Alviano, Dodaro, and Ricca, 2015]

Open problem: achieve a better understand of the impact of these new variables on the SAT solving process

Dealing with Weighted Soft Clauses

Presented algorithms using unit weight soft clauses

How do we deal with clauses of different weights!

Clause Cloning

- Methods can deal with new core when soft clauses in core have same weight.
- If this weight is w:

  $k = k + w$ rather than $k = k + 1$.
- Clause Cloning is the method used to deal with varying weights.

[Ansótegui, Bonet, and Levy, 2009; Manquinho, Silva, and Planes, 2009]

Clause Cloning

1. K is new core.
2. $w_{\min}$ is minimum weight in K.
3. We split each clause $(c, w) \in K$ into two clauses
   (1) $(c, w_{\min})$ (2) $(c, w - w_{\min})$.
4. Keep all clauses (2) $(c, w - w_{\min})$ as soft clauses
   (discard zero weight clauses)
5. We let K be all clauses (1) $(c, w_{\min})$
6. We process K as a new core
   (all clauses in K have the same weight)

Sat Based MaxSat: Summary

- Techniques are effective on large MaxSat problems, especially those with many hard clauses.
- The innovation is in obtaining more efficient ways to encode and solve the individual SAT decision problems that have to be solved.
- Some work done on understand the core structure and its impact on SAT solving efficiency but more needed.

[Bacchus and Narodytska, 2014]

- The method of clause cloning for dealing with varying clause weights is not effective when there are many different weights.

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 H is a *hitting set* of $\mathcal{S}$ if $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.

A hitting set H is *optimal* if no $H' \subset \bigcup \mathcal{S}$ with $|H'| < |H|$ 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 H is a *hitting set* of $\mathcal{S}$ if $H \cap S \neq \emptyset$ for all $S \in \mathcal{S}$.

A hitting set H is *optimal* if no $H' \subset \bigcup \mathcal{S}$ with $|H'| < |H|$ 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 H of the *set of UNSAT cores of* F,
there is an optimal solutions τ to F such that τ satisfies exactly the clauses $F \setminus H$.

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 F of the UNSAT cores of F.
  - Implicit refers to not necessarily having all MUSes of F.
- Find a solution to $F \setminus H$.

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 H over $\mathcal{K}$,
  and *rule out the clauses in H for the next SAT solver call*

  using an IP solver

...until the SAT solver returns 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 H over $\mathcal{K}$,
  and *rule out the clauses in H for the next SAT solver call*

  using an IP solver

...until the SAT solver returns satisfying assignment.

Hitting Set Problem as Integer Programming

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

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

- $b_C = 1$ 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

Instantiation of the implicit hitting set approach

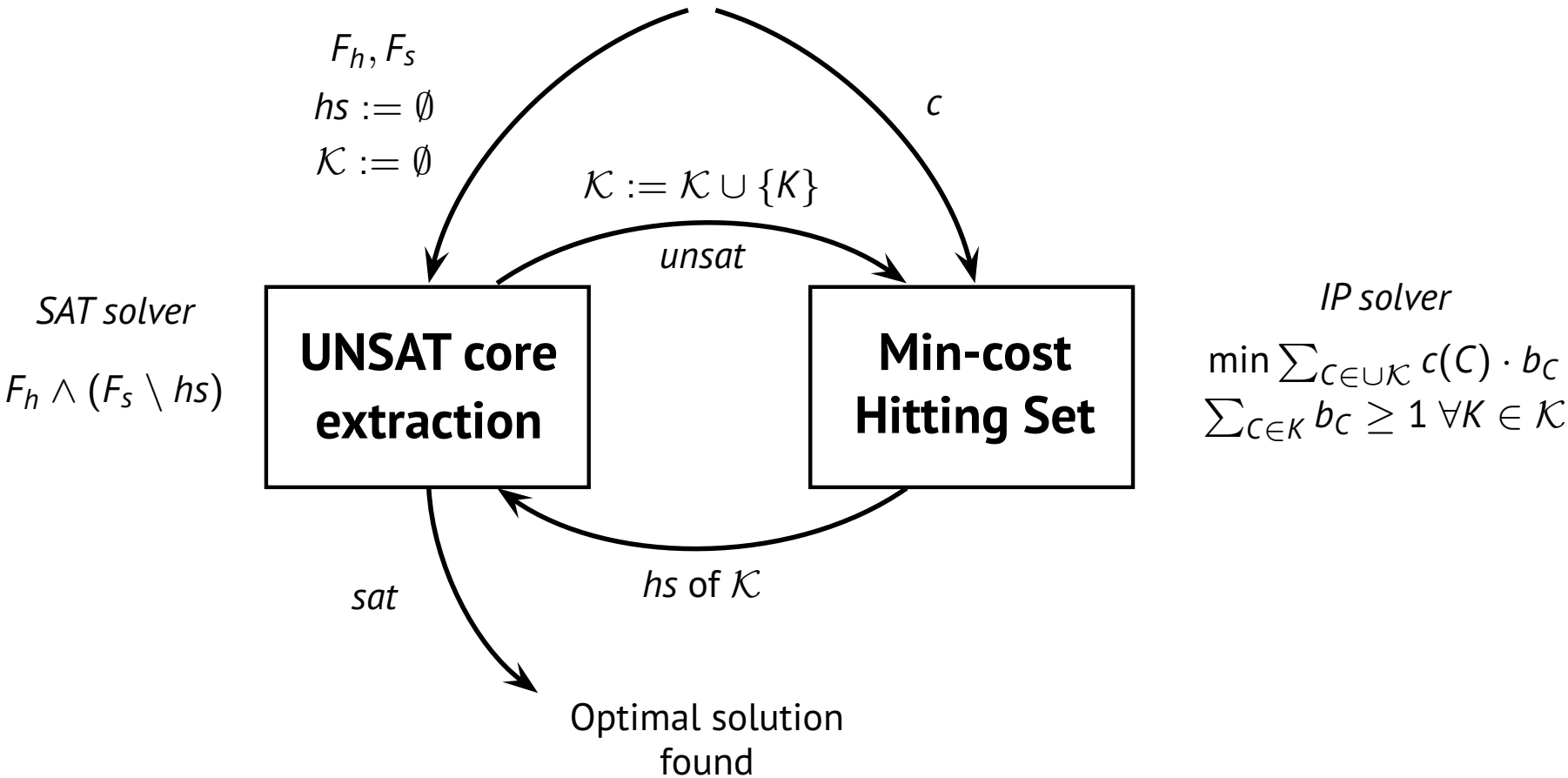
[Moreno-Centeno and Karp, 2013]

- Also possible to instantiate beyond MaxSat

[Saikko, Wallner, and Järvisalo, 2016]

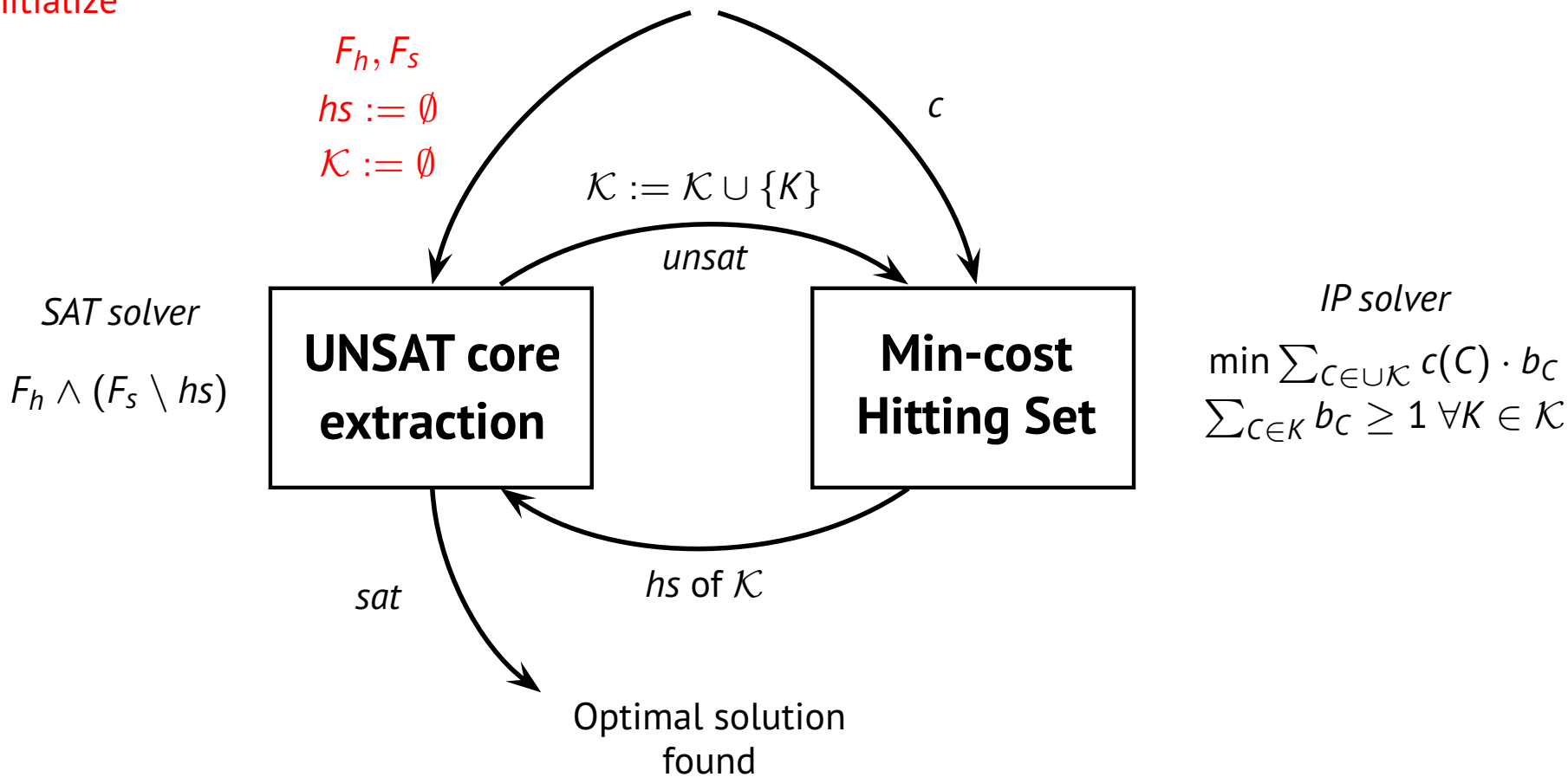
Solving MaxSat by SAT and Hitting Set Computations

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



Solving MaxSat by SAT and Hitting Set Computations

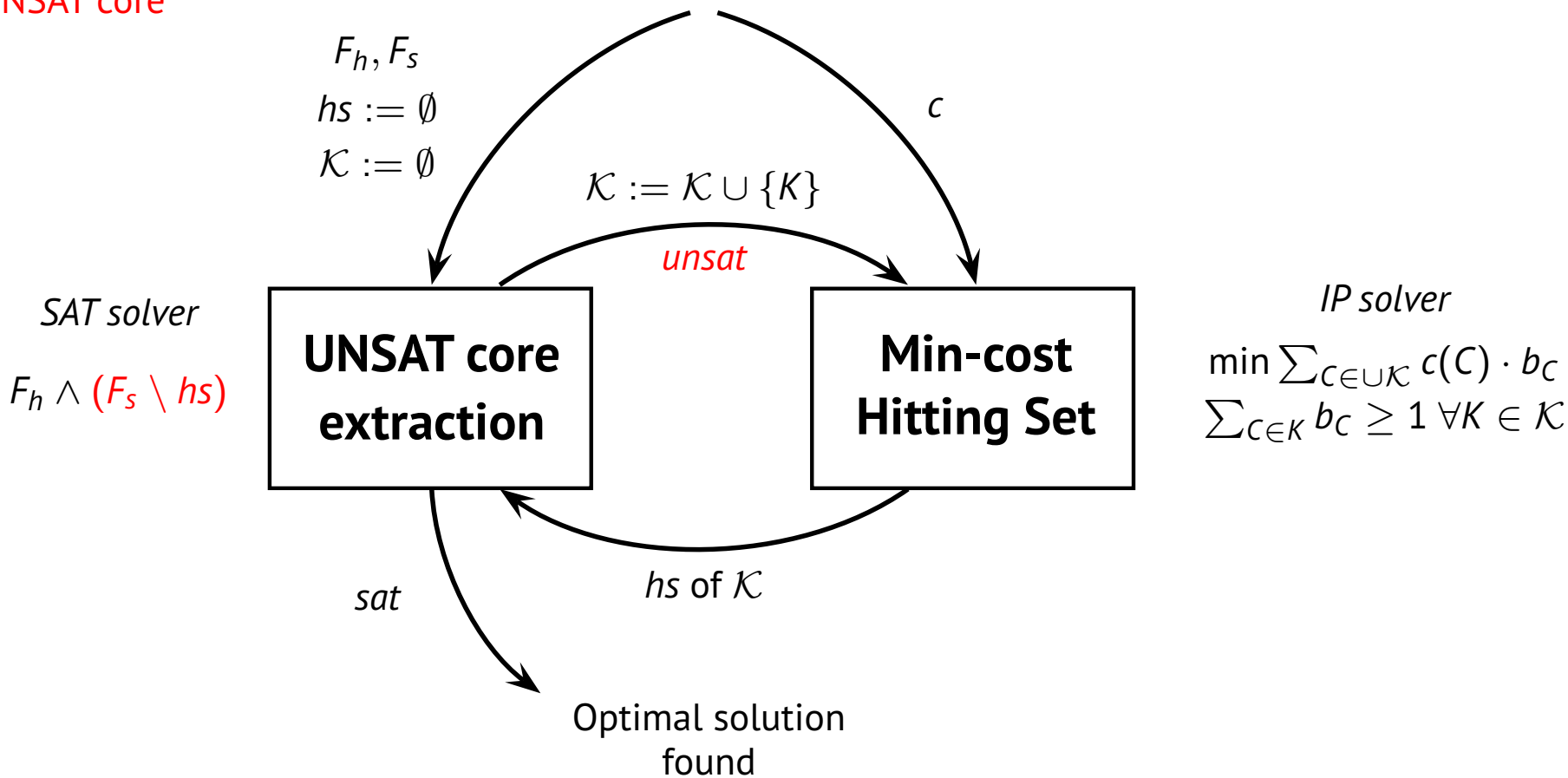
Input:
hard clauses F_h, soft clauses F_s, weight function $c : F_s \mapsto \mathbb{R}^+$



Solving MaxSat by SAT and Hitting Set Computations

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

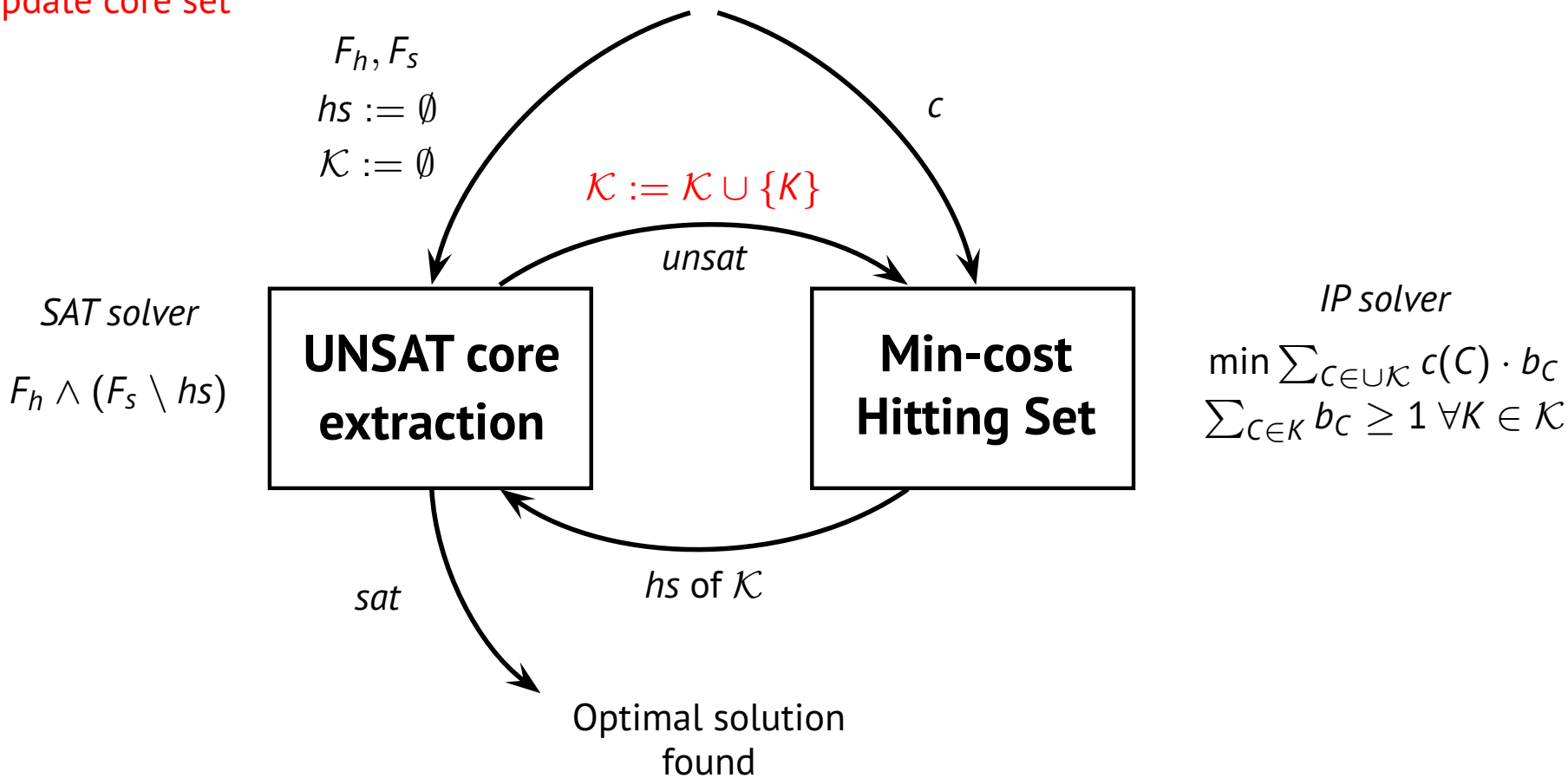
2. UNSAT core



Solving MaxSat by SAT and Hitting Set Computations

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

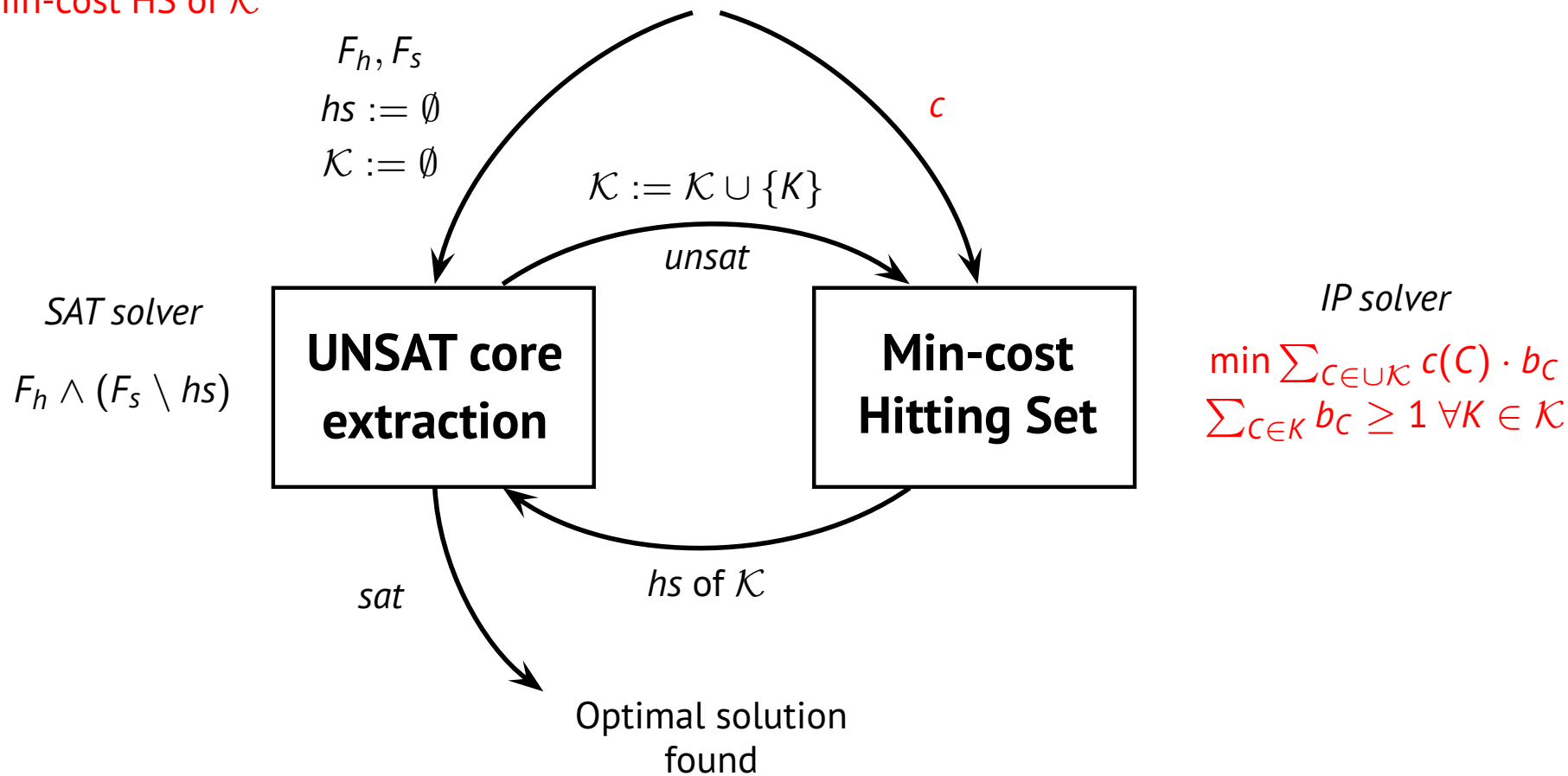
3. Update core set



Solving MaxSat by SAT and Hitting Set Computations

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

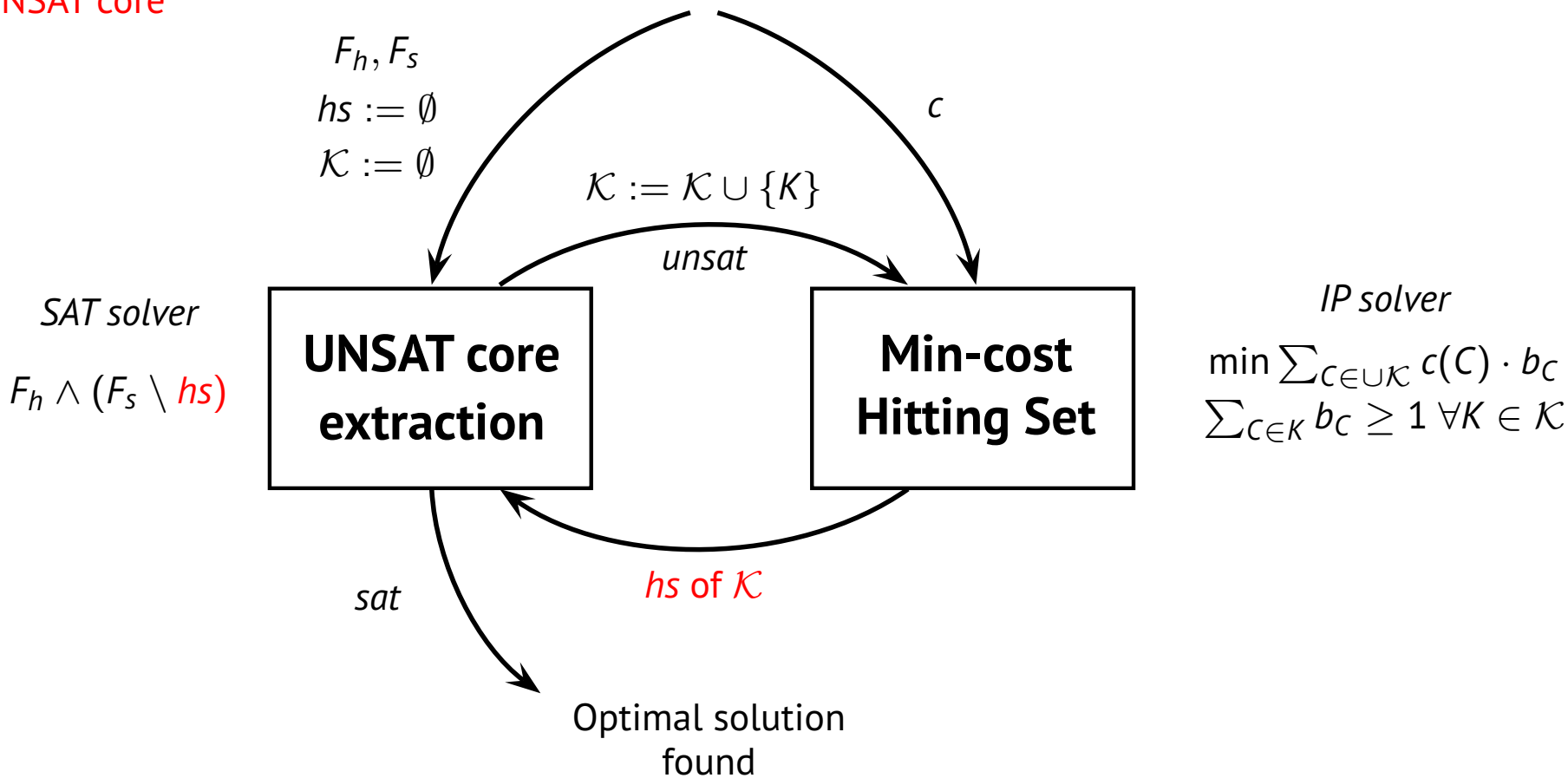
4. Min-cost HS of $\mathcal{K}$



Solving MaxSat by SAT and Hitting Set Computations

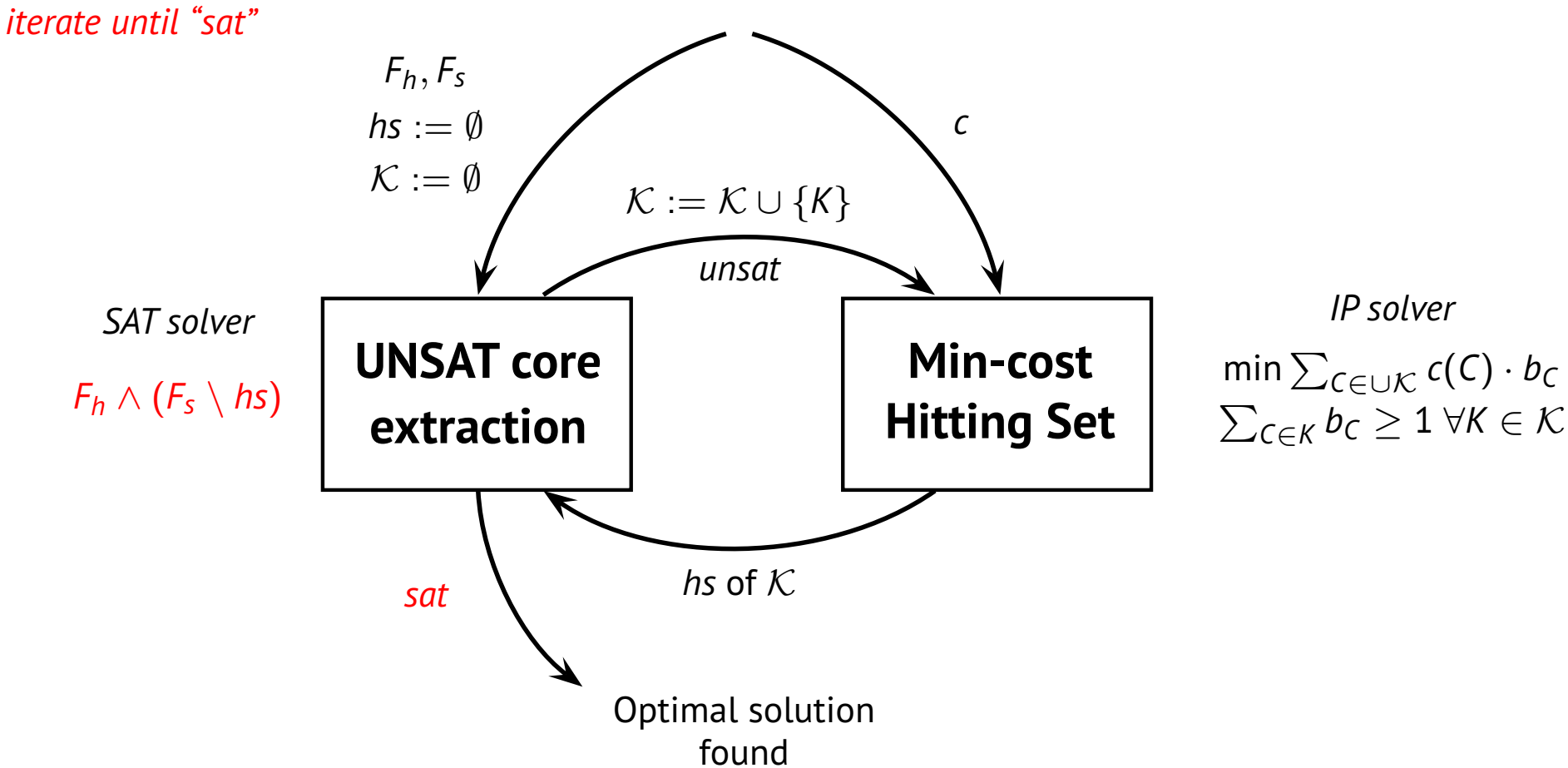
Input:
hard clauses F_h, soft clauses F_s, weight function $c : F_s \mapsto \mathbb{R}^+$

5. UNSAT core



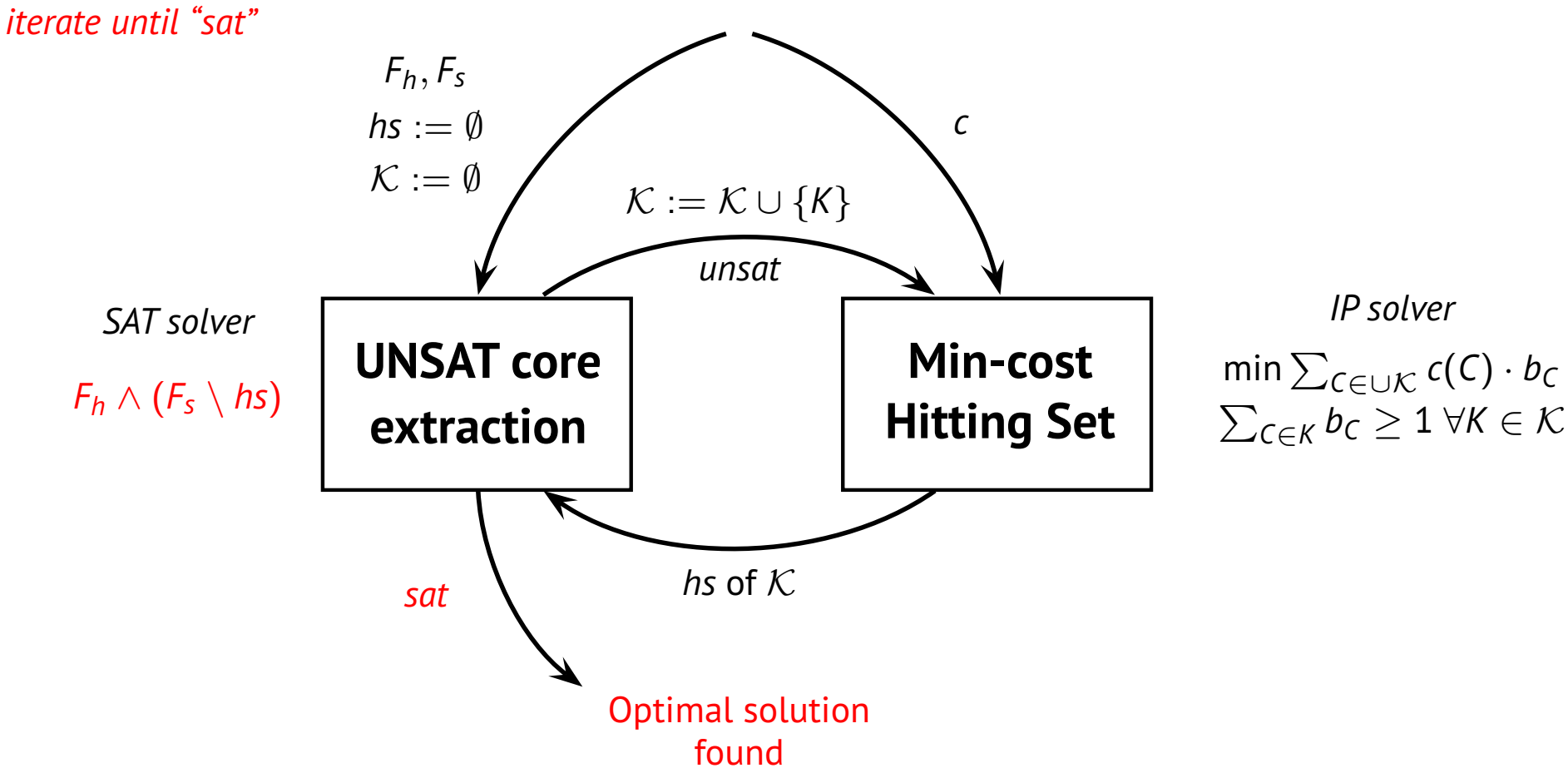
Solving MaxSat by SAT and Hitting Set Computations

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



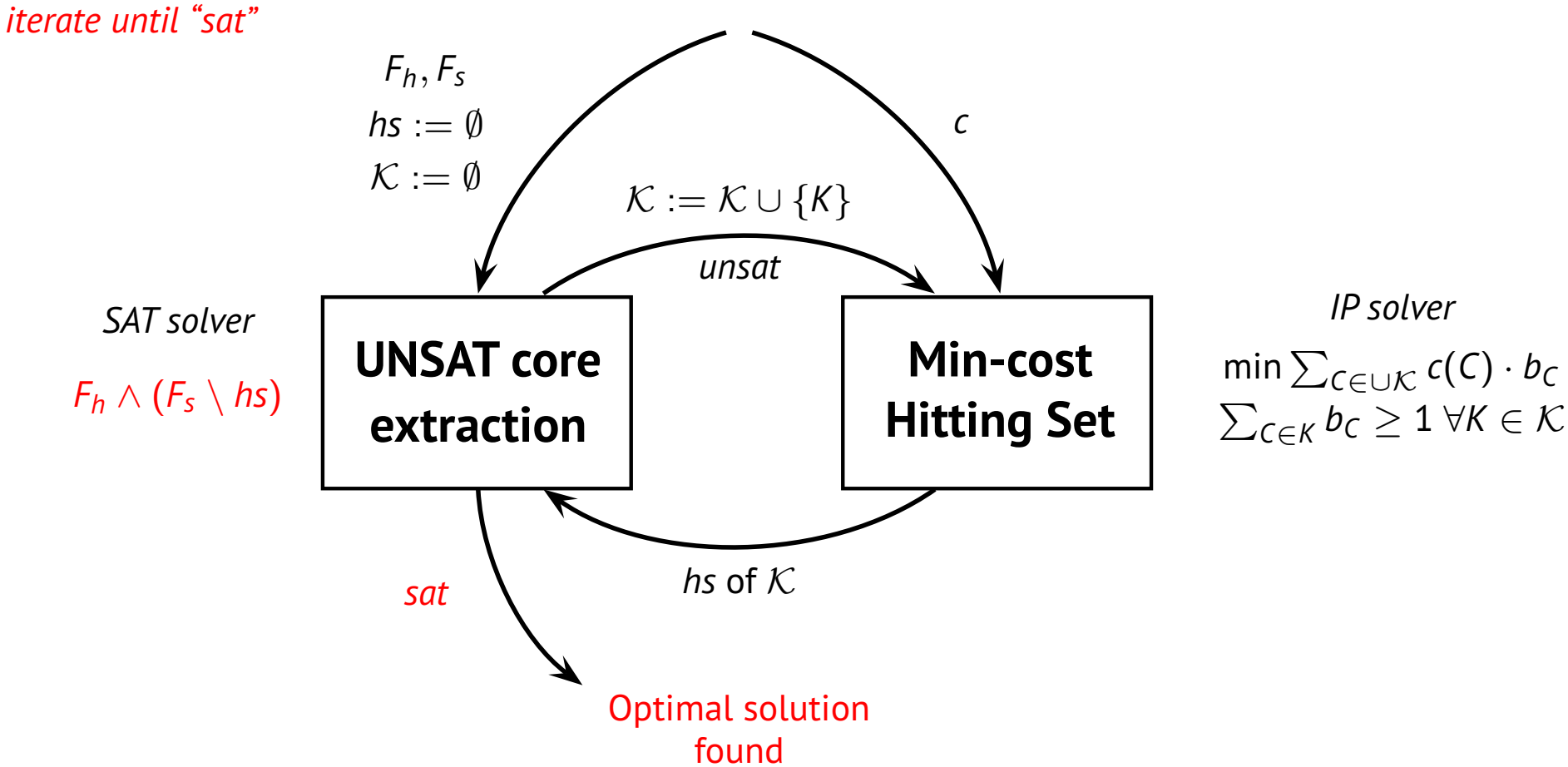
Solving MaxSat by SAT and Hitting Set Computations

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



Solving MaxSat by SAT and Hitting Set Computations

Intuition: After optimally hitting *all* cores of $F_h \wedge F_s$ by *hs*:
any solution to $F_h \wedge (F_s \setminus hs)$ *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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \wedge (F_s \setminus \emptyset)$

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

- 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

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

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

- 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 $F_h \wedge (F_s \setminus \{C_1\})$

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 $F_h \wedge (F_s \setminus \{C_1\}) \rightsquigarrow$ UNSAT core $K = \{C_9, C_{10}, C_{11}, C_{12}\}$

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

- 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

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

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

- SAT solve $F_h \wedge (F_s \setminus \{C_1, C_9\})$

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 $F_h \wedge (F_s \setminus \{C_1, C_9\})$
  $\rightsquigarrow$ UNSAT core $K = \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\}$

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \wedge (F_s \setminus \{C_4, C_9\})$

MaxSat by SAT and Hitting Set Computation: Example

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

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

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

MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve $F_h \wedge (F_s \setminus \{C_4, C_9\}) \rightsquigarrow$ 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
- ...

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

For more on some of the details, see [Davies and Bacchus, 2011, 2013b,a]

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.
- On problems with no weights or very few weights can be outperformed by SAT based approaches.

Iterative Use of SAT Solvers (for MaxSat)

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"
- Various modern CDCL SAT solvers implement an API for solving under assumption

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls

Iterative Use of SAT Solvers (for MaxSat)

- In many application scenarios, including MaxSat:
  it is beneficial to be able to make several SAT checks on the *same* input CNF formula under different forced partial assignments.
  - Such forced partial assignments are called *assumptions*
  - "Is the formula F satisfiable under the assumption $x = 1$?"
- Various modern CDCL SAT solvers implement an API for solving under assumption
  - The input formula is read in only once
  - The user implements a iterative loop that calls the same solver instantiation under different sets of assumptions
  - The calls can be adaptive, i.e., assumptions of future SAT solver calls can depend on the results of the previous solver calls
  - *The solver can keep its internal state from the previous solver call to the next*
    - Learned clauses
    - Heuristic scores

Incremental APIs in SAT Solver: Minisat

Minisat `http://www.minisat.se`

- Perhaps the most used SAT solver
- Implements the CDCL algorithm
- Very clean and easy-to-understand-and-modify source code
- Offers an incremental interface
  - `solve`(partial assignment: list of assumptions):
    for making a SAT solver call under a set of assumptions
  - `analyzeFinal`:
    returns an explanation for unsatisfiability under the assumptions as a clause over a subset of the assumptions
  - `addClauses`:
    for adding more clauses between solver calls

Explaining Unsatisfiability

CDCL SAT solvers determine unsatisfiability when learning the empty clause

- By propagating a conflict at decision level 0

Explaining unsatisfiability under assumptions

- The reason for unsatisfiability can be traced back to assumptions that were necessary for propagating the conflict at level 0.
- Essentially:
  - Force the assumptions as the first “decisions”
  - When one of these decisions results in a conflict: trace the reason of the conflict back to the forced assumptions

Implementing MaxSat Algorithms via Assumptions

- Instrument each soft clause C_i with a new "assumption" variable a_i
  $\rightsquigarrow$ replace C_i with $(C_i \vee a_i)$ for each soft clause C_i
- $a_i = 0$ switches C_i "on",
  $a_i = 1$ switches C_i "off"
- MaxSat core: a subset of the assumptions variables a_is
  - Heavily used in *core-based* MaxSat algorithms
  - In the *implicit hitting set approach:*
    hitting sets over sets of assumption variables
  - Cost of including a_i in a core (i.e., assigning $a_i = 1$):
    weight of the soft clause C_i
- Can state cardinality constraints directly over the assumption variables
  - Heavily used in MaxSat algorithms employing cardinality constraints

Modelling and Applications

Representing High-Level Soft Constraints in MaxSat

MaxSat allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

Representing High-Level Soft Constraints in MaxSat

MaxSat allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

Basic Idea

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

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

Softening 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: (a) with weight $W_{\mathcal{C}}$.

Representing High-Level Soft Constraints in MaxSat

MaxSat allows for compactly encoding various types of high-level finite-domain soft constraints

- Due to Cook-Levin Theorem:
  Any NP constraint can be polynomially represented as clauses

Basic Idea

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

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

Softening 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: (a) with weight $W_{\mathcal{C}}$.

Important for various applications of MaxSat

MaxSat Applications

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]
...	

MaxSat Applications

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	[Wallner, Niskanen, and Järvisalo, 2016]
...	

MaxSat Applications

probabilistic inference	[Park, 2002]
design debugging	[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
	[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
maximum quartet consistency	[Morgado and Marques-Silva, 2010]
software package management	[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
	[Ignatiev, Janota, and Marques-Silva, 2014]
Max-Clique	[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
fault localization	[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
restoring CSP consistency	[Lynce and Marques-Silva, 2011]
reasoning over bionetworks	[Guerra and Lynce, 2012]
MCS enumeration	[Morgado, Liffiton, and Marques-Silva, 2012]
heuristics for cost-optimal planning	[Zhang and Bacchus, 2012]
optimal covering arrays	[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
correlation clustering	[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
treewidth computation	[Berg and Järvisalo, 2014]
Bayesian network structure learning	[Berg, Järvisalo, and Malone, 2014]
causal discovery	[Hyttinen, Eberhardt, and Järvisalo, 2014]
visualization	[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
model-based diagnosis	[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
cutting planes for IPs	[Saikko, Malone, and Järvisalo, 2015]
argumentation dynamics	**[Wallner, Niskanen, and Järvisalo, 2016]**
...	

MaxSat-based Correlation Clustering

[Berg and Järvisalo, 2016]

Correlation Clustering

[Bansal, Blum, and Chawla, 2004]

Partitioning data points into *clusters* based on pair-wise *similarity information*

- NP-hard optimization problem [Bansal, Blum, and Chawla, 2004]
- The number of clusters available *not fixed*
  - Intuitively: objective function under minimization aims at balancing *precision* and *recall*
- Several approximation algorithms proposed

  [Bansal, Blum, and Chawla, 2004; Ailon, Charikar, and Newman, 2008; Charikar, Guruswami, and Wirth, 2005; Demaine, Emanuel, Fiat, and Immorlica, 2006]
  - Approximation guarantees under binary similarity information
  - Semi-definite relaxation, quadratic programming

Correlation Clustering

[Bansal, Blum, and Chawla, 2004]

Applications in various settings

- Clustering documents based on topics

  [Bansal, Blum, and Chawla, 2004; Gael and Zhu, 2007]

- Biosciences [Ben-Dor, Shamir, and Yakhini, 1999]
- Social network analysis, information retrieval

  [Bonchi, Gionis, and Ukkonen, 2011; Bonchi, Gionis, Gullo, and Ukkonen, 2012; Cesa-Bianchi, Gentile, Vitale, and Zappella, 2012]

- Consensus clustering [Bonizzoni, Vedova, Dondi, and Jiang, 2005]
  for e.g. microarray data analysis

  [Filkov and Skiena, 2004b,a; Giancarlo and Utro, 2011; Yu, Wong, and Wang, 2007]

Cost-Optimal Correlation Clustering

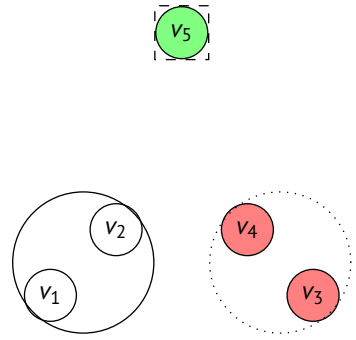
V	f_1	f_2	f_3	...
v_1	0.5	1	3	...
v_2	-3	0	-2	...
v_3	0.7	1	5	...
v_4	4	1	7	...
v_5	6	0	10	...

DATA

$$\Rightarrow \quad W = \begin{bmatrix} 0 & 1 & 0.7 & 0 & 0.2 \\ 1 & 0 & 4 & -7 & -5 \\ 0.7 & 4 & 0 & \infty & 0 \\ 0 & -7 & \infty & 0 & -3 \\ 0.2 & -5 & 0 & -3 & 0 \end{bmatrix}$$

SIMILARITY MATRIX

$\Rightarrow$ MAXSAT: encoding + solving $\Rightarrow$



SOLUTION CLUSTERING

INPUT: a similarity matrix W,

TASK: find a cost-optimal correlation clustering, i.e., a function $cl^* : V \to \mathbb{N}$ minimizing

$$\min_{cl:\, V \to \mathbb{N}} \sum_{\substack{cl(v_i)=cl(v_j) \\ i<j}} (\mathcal{I}[-\infty < W(i,j) < 0] \cdot |W(i,j)|) +$$

$$\sum_{\substack{cl(v_i)\neq cl(v_j) \\ i<j}} (\mathcal{I}[\infty > W(i,j) > 0] \cdot W(i,j))$$

where the indicator function $\mathcal{I}[b] = 1$ iff the condition b is true.

Why MaxSat-based Correlation Clustering?

- Cost-optimal solutions *notably* better (w.r.t. objective) compared to previous approximation algorithms
  - Both semi-definite relaxations and specialized algorithms with approximation guarantees
- Allows for *constrained* correlation clustering [Wagstaff and Cardie, 2000; Wagstaff, Cardie, Rogers, and Schrödl, 2001; Davidson and Ravi, 2007]

  Can adapt to additional user knowledge simply via imposing additional clauses
  - No need to adapt search algorithm — not always the case for approximation algorithms
- MaxSat (using the implicit hitting set approach) scales better than IP solvers

In this tutorial: to illustrate bit-level "log encodings"

How to encode non-binary variables with large domains

Correlation Clustering as an Integer Program

[Ailon, Charikar, and Newman, 2008; Gael and Zhu, 2007]

- Use indicator variables $x_{ij} \in \{0, 1\}$.
- $x_{ij} = 1$ iff $cl(i) = cl(j)$, i.e., points i and j co-clustered

IP formulation

$$\text{Minimize} \quad \sum_{\substack{-\infty < W(i,j) < 0 \\ i<j}} \left(x_{ij} \cdot |W(i,j)|\right) - \sum_{\substack{\infty > W(i,j) > 0 \\ i<j}} \left(x_{ij} \cdot W(i,j)\right)$$

$$\begin{aligned} \text{where} \quad x_{ij} + x_{jk} &\leq 1 + x_{ik} && \text{for all distinct } i, j, k \\ x_{ij} &= 1 && \text{for all } W(i,j) = \infty \\ x_{ij} &= 0 && \text{for all } W(i,j) = -\infty \\ x_{ij} &\in \{0, 1\} && \text{for all } i, j \end{aligned}$$

Transitivity-based encoding

$\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ constraints *very large*

Reformulating the IP as MaxSat

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ clauses.
- Same indicator variables: $x_{ij} = 1$ iff $cl(v_i) = cl(v_j)$

Reformulating the IP as MaxSat

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ clauses.
- Same indicator variables: $x_{ij} = 1$ iff $cl(v_i) = cl(v_j)$

Hard clauses

Encoding the linear constraint $x_{ij} + x_{jk} \leq 1 + x_{ik}$:

- $(x_{ij} \wedge x_{jk}) \rightarrow x_{ik}$
  as clause: $(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})$

Encoding $W(i,j) = \infty$: $\quad (x_{ij})$

Encoding $W(i,j) = -\infty$: $\quad (\neg x_{ij})$

Reformulating the IP as MaxSat

- Hard clauses encode well-defined clusterings
- Soft clauses encode the object function
- $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ clauses.
- Same indicator variables: $x_{ij} = 1$ iff $cl(v_i) = cl(v_j)$

Hard clauses

Encoding the linear constraint $x_{ij} + x_{jk} \leq 1 + x_{ik}$:

- $(x_{ij} \wedge x_{jk}) \rightarrow x_{ik}$
  as clause: $(\neg x_{ij} \vee \neg x_{jk} \vee x_{ik})$

Encoding $W(i,j) = \infty$: $\quad (x_{ij})$

Encoding $W(i,j) = -\infty$: $\quad (\neg x_{ij})$

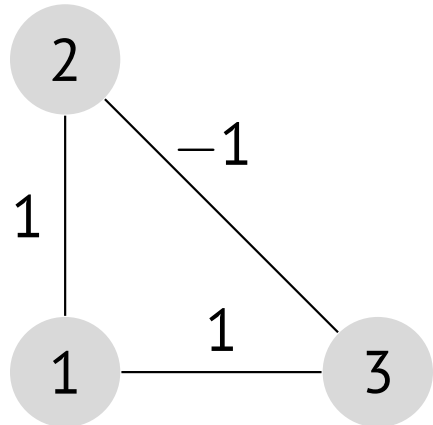
Soft clauses

Encoding the objective function:

- For $W(i,j) \in (0, \infty)$: $\quad (x_{ij})$ with weight $W(i,j)$
- For $W(i,j) \in (-\infty, 0)$: $\quad (\neg x_{ij})$ with weight $|W(i,j)|$

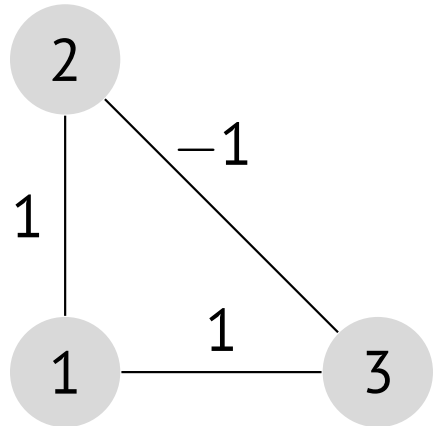
Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$

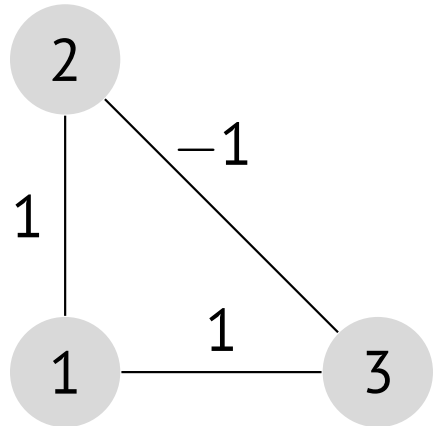


- Hard clauses:

$$\{(\neg x_{12} \vee \neg x_{23} \vee x_{13}), (\neg x_{12} \vee \neg x_{13} \vee x_{23}), (\neg x_{23} \vee \neg x_{13} \vee x_{12})\}$$

Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



- Hard clauses:

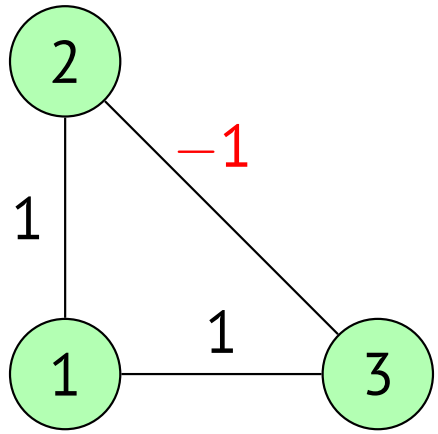
$$\{(\neg x_{12} \vee \neg x_{23} \vee x_{13}), (\neg x_{12} \vee \neg x_{13} \vee x_{23}), (\neg x_{23} \vee \neg x_{13} \vee x_{12})\}$$

- Soft clauses:

$$\{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\}$$

Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



- Hard clauses:

$$\{(\neg x_{12} \vee \neg x_{23} \vee x_{13}), (\neg x_{12} \vee \neg x_{13} \vee x_{23}), (\neg x_{23} \vee \neg x_{13} \vee x_{12})\}$$

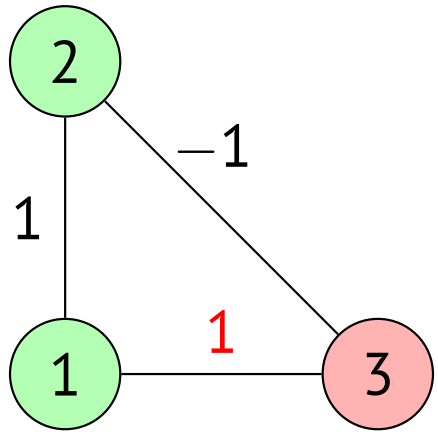
- Soft clauses:

$$\{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\}$$

- 1 Cluster. Cost: 1.

Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



- Hard clauses:

$$\{(\neg x_{12} \vee \neg x_{23} \vee x_{13}), (\neg x_{12} \vee \neg x_{13} \vee x_{23}), (\neg x_{23} \vee \neg x_{13} \vee x_{12})\}$$

- Soft clauses:

$$\{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\}$$

- 2 Clusters. Cost: 1.

A More Compact MaxSat Encoding

Bit-level / log encodings

For representing non-binary variables with large domains

- To represent the value assignment of a variable with domain $D = \{0, \ldots, |D| - 1\}$:
  - use $\log |D|$ Boolean variables $b_1..b_{\log |D|}$
  - Interpret an assignment to $b_1..b_{\log D}$ as the bit-representation of a value in D.

Does not always pay of due to poor propagation properties!

However, in correlation clustering:

- Domain-size: number of clusters
- Can be up to number of points to be clustered
- For example: the cluster assignment of each of 512 points can be represented with $\log_2 512 = 9$ bits

Log Encoding of Correlation Clustering

Variables

- Cluster assignment of point i: variables b_i^k for $k = 1..\log N$.
- $S_{ij} = 1$ iff points i and j are co-clustered
- Auxiliary: $EQ_{ij}^k = 1$ iff $b_i^k = b_j^k$

Hard clauses

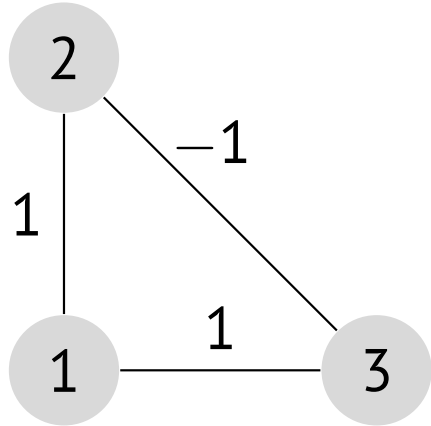
- Semantics of EQ_{ij}^k: $\quad EQ_{ij}^k \leftrightarrow (b_i^k \leftrightarrow b_j^k)$
- Semantics of S_{ij}: $\quad S_{ij} \leftrightarrow (EQ_{ij}^1 \wedge \cdots \wedge EQ_{ij}^{\log N})$
- Encoding $W(i,j) = \infty$: $\quad (S_{ij})$
  Encoding $W(i,j) = -\infty$: $\quad (\neg S_{ij})$

Soft clauses

- For $W(i,j) \in (0, \infty)$: $\quad (S_{ij})$ with weight $W(i,j)$
- For $W(i,j) \in (-\infty, 0)$: $\quad (\neg S_{ij})$ with weight $|W(i,j)|$

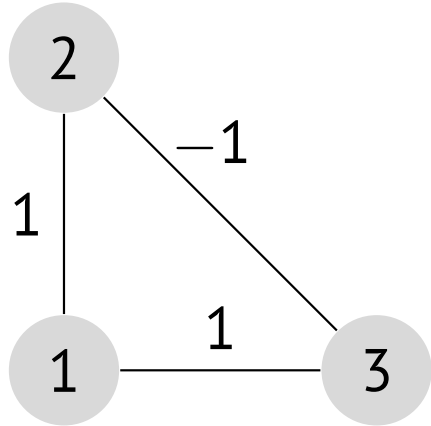
Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



- Hard clauses:

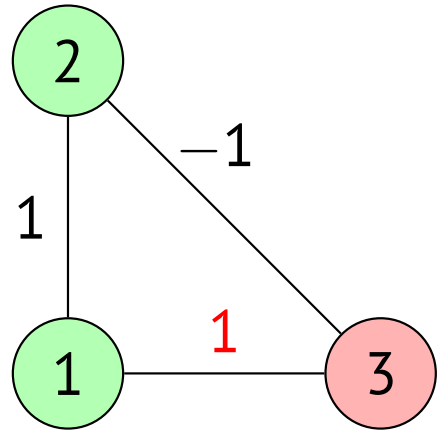
$$S_{12} \leftrightarrow (EQ_{12}^1 \wedge EQ_{12}^2) \qquad EQ_{12}^1 \leftrightarrow (b_1^1 \leftrightarrow b_2^1)$$
$$EQ_{12}^2 \leftrightarrow (b_1^2 \leftrightarrow b_2^2)$$
$$S_{13} \leftrightarrow (EQ_{13}^1 \wedge EQ_{13}^2) \qquad EQ_{13}^1 \leftrightarrow (b_1^1 \leftrightarrow b_3^1)$$
$$EQ_{13}^2 \leftrightarrow (b_1^2 \leftrightarrow b_3^2)$$
$$S_{23} \leftrightarrow (EQ_{23}^1 \wedge EQ_{23}^2) \qquad EQ_{23}^1 \leftrightarrow (b_2^1 \leftrightarrow b_3^1)$$
$$EQ_{23}^2 \leftrightarrow (b_1^2 \leftrightarrow b_3^2)$$

- Soft clauses:

$$\{(S_{12}; 1), (S_{13}; 1), (\neg S_{23}; 1)\}$$

Example

$$W = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & -1 \\ 1 & -1 & 0 \end{bmatrix}$$



Clustering:
points 1,2 in cluster 1
point 3 in cluster 2

- Hard clauses:

$$S_{12} \leftrightarrow (EQ^1_{12} \wedge EQ^2_{12}) \qquad EQ^1_{12} \leftrightarrow (b^1_1 \leftrightarrow b^1_2)$$
$$EQ^2_{12} \leftrightarrow (b^2_1 \leftrightarrow b^2_2)$$
$$S_{13} \leftrightarrow (EQ^1_{13} \wedge EQ^2_{13}) \qquad EQ^1_{13} \leftrightarrow (b^1_1 \leftrightarrow b^1_3)$$
$$EQ^2_{13} \leftrightarrow (b^2_1 \leftrightarrow b^2_3)$$
$$S_{23} \leftrightarrow (EQ^1_{23} \wedge EQ^2_{23}) \qquad EQ^1_{23} \leftrightarrow (b^1_2 \leftrightarrow b^1_3)$$
$$EQ^2_{23} \leftrightarrow (b^2_1 \leftrightarrow b^2_3)$$

- Soft clauses:

$$\{(S_{12}; 1), (S_{13}; 1), (\neg S_{23}; 1)\}$$

Experiments

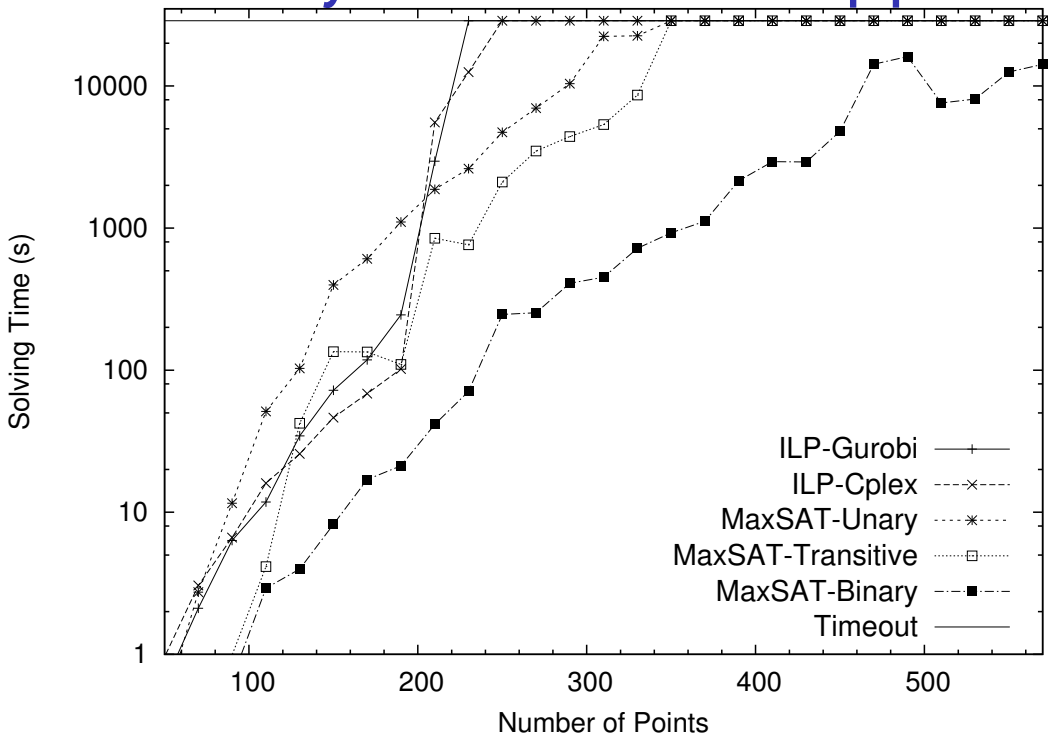
MaxSat solver: MaxHS (implicit hitting set approach)

Protein sequencing data: similarity information over amino-acid sequences

Compared with:

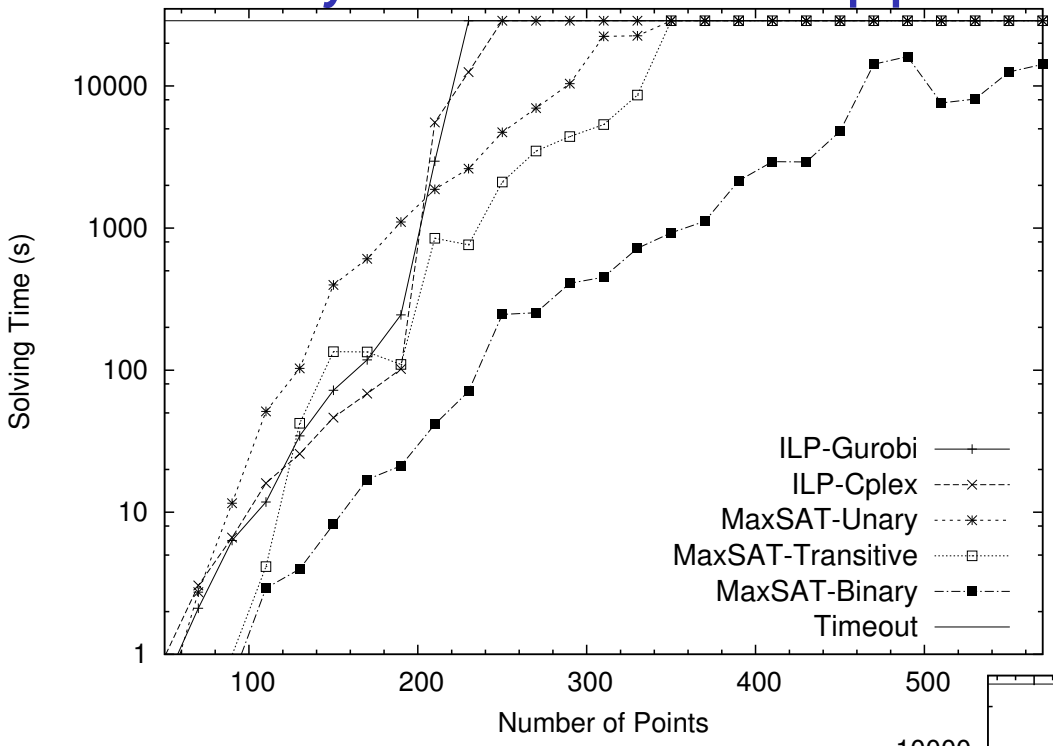
- Exact state-of-the-art IP solvers:
  CPLEX, Gurobi
- Approximation algorithms for correlation clustering:
  KwickCluster (KC), SDPC (semi-definite relaxation of the IP)
- SCPS: a dedicated spectral clustering algorithms for *the specific type of data*

Scalability of the Exact Approaches



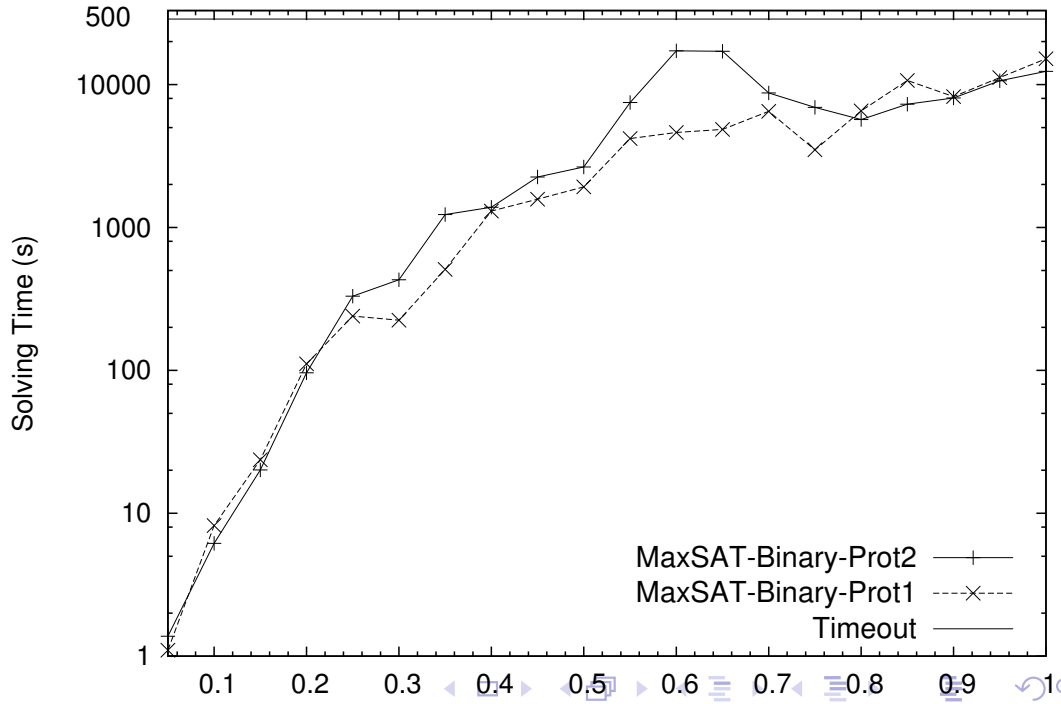
- Log encoding scales further w.r.t. number of datapoints considered

Scalability of the Exact Approaches

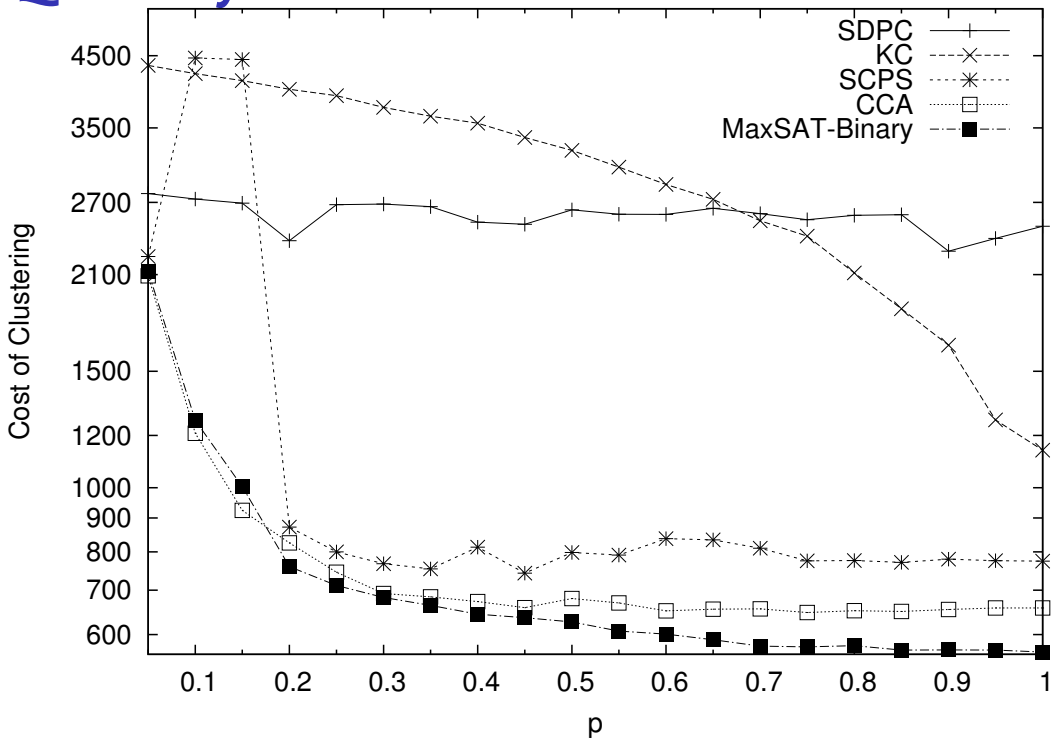


- Log encoding scales further w.r.t. number of datapoints considered

- Scalability under incomplete similarity information
- (IP does not scale up to the full set of point)

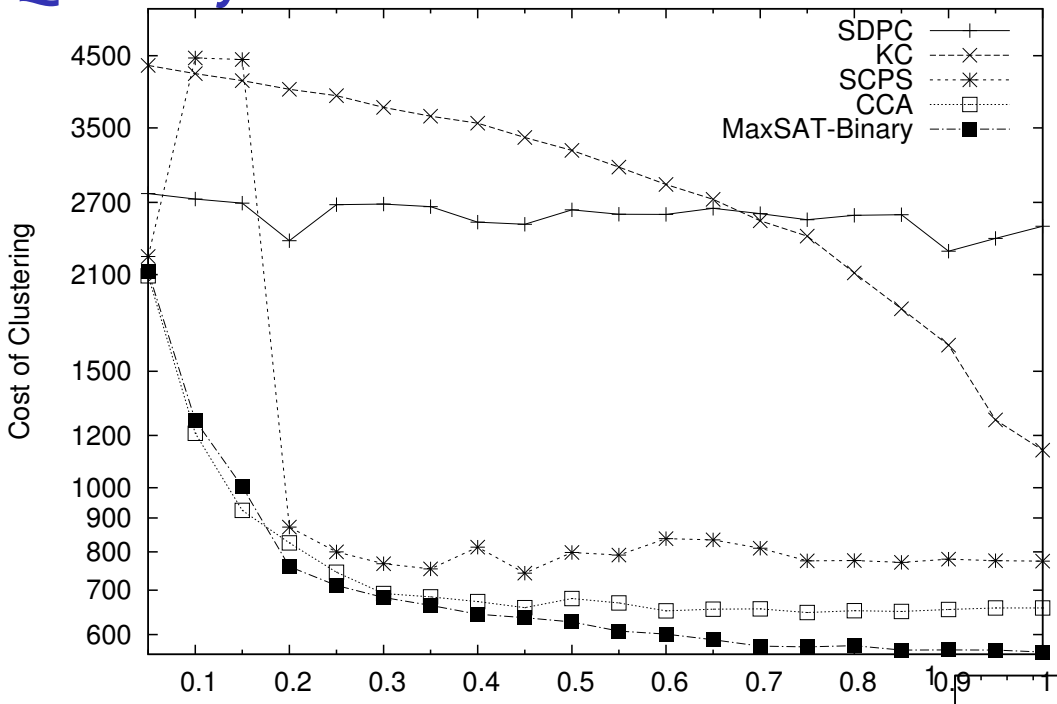


Quality of Solutions



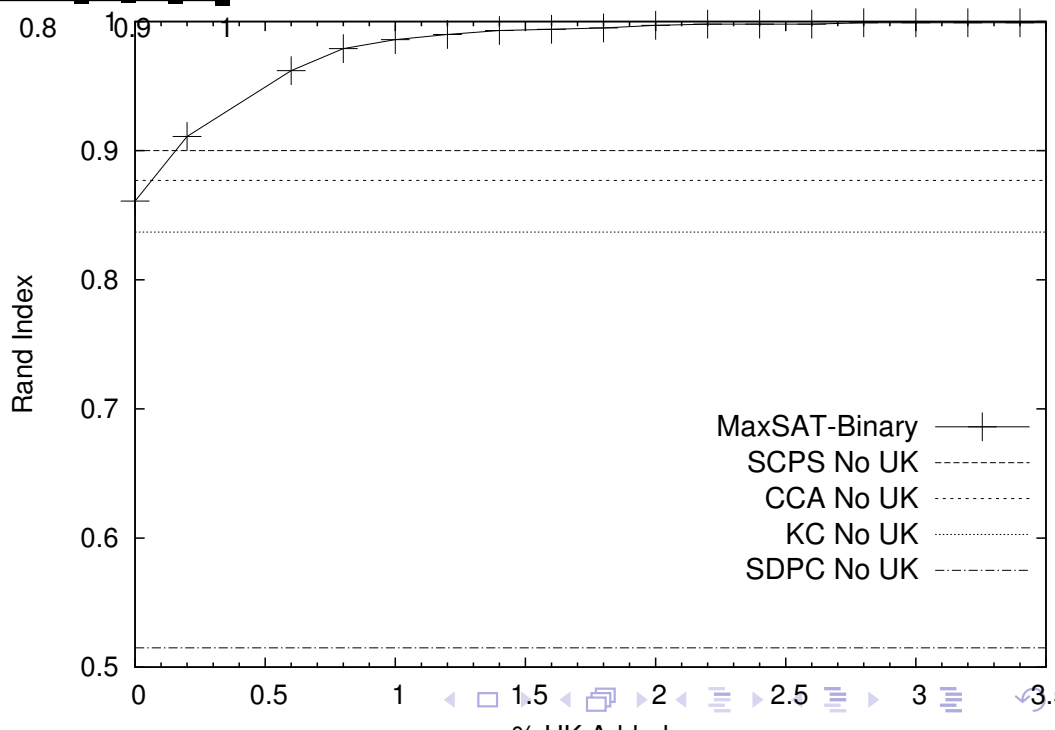
- Notably better solution costs,
  esp. on incomplete similarity info
- Consistently better than the data-specific SCPS

Quality of Solutions



- Notably better solution costs,
  esp. on incomplete similarity info
- Consistently better than the data-specific SCPS

- Rand index: typical clustering quality measure
- user knowledge (UK) on a golden clustering: Rand index for MaxSat goes quickly beyond the others



Heuristics for Planning using MaxSat

[Zhang and Bacchus, 2012]

Cost Optimal Planning

- An important extension of the classical planning problem is the cost optimal classical planning problem.
- Classical planning involves full information: the state is fully-known as are the effects and preconditions of all actions.
- Want to transform the initial state into a state satisfying the goal with a minimum cost sequence of actions.

Planning Formalism

- Planning problem $\langle P, I, G, A, aCost \rangle$
  1. Set of facts P (propositions)
  2. A state S is a subset of P (a set of true facts)
     Closed World: If $p \in P$ is not in S then p is false in S.
  3. Initial state $I \subseteq P$
  4. Goal condition $G \subseteq P$
     no Closed world for G: $p \notin G \not\rightarrow \neg p$
  5. Actions A
  6. Action cost function $aCost$: $a \in A$ incurs cost $aCost(a) > 0$.

Actions

- Each action $a \in A$ is $\langle pre(a), add(a), del(a) \rangle$
- Preconditions $pre(a)$; add effects $add(a)$, and $del(a)$ delete effects.
- Actions map states S to new states $a(S)$:
  1. If $pre(a) \not\subseteq S$ we can't apply a to S (a is not executable in S).
  2. If a is executable in S then

$$a(S) = S - del(a) \cup add(a).$$

Plans

- A plan $\Pi = \langle a_1, a_2, \ldots, a_n \rangle$ for a planning problem $\langle P, I, G, A, aCost \rangle$ is a sequence of actions from A such that:
  1. $S_0 = I, S_1 = a_1(S_0), \ldots, S_n = a_n(S_{n-1})$
  2. Each a_i is executable in S_{i-1}
  3. $G \subseteq S_n$.
- Each action is applied in sequence. The actions sequence must be executable, and the final state must satisfy the goal.

Cost Optimal Plans

- The cost of a plan Π is the sum of $aCost(a_i)$ for all $a_i \in \Pi$.
- Given a planning problem we want to find a plan Π for it with minimum cost.

Complexity

- Classical planning is PSPACE-complete, i.e., beyond the complexity of MaxSat.
- This complexity arises from not knowing apriori the number of actions in a plan.
- To find an optimal cost plan we would have to impose a bound on the number of actions in the plan, and iterate on this bound.
- The iteration is not simple, as when actions have varying costs, a longer plan might be cheaper.
- One approach for directly using MaxSat was suggested by Robinson et al. [Robinson, Gretton, Pham, and Sattar, 2010]

Heuristic Search

- The most successful approach to solving planning problems is heuristic search.
- When an admissible heuristic is used search becomes A*-search and we can find optimal plans.
- Now the problem becomes what is a good heuristic to use.
- A great deal of research has been done on this problem
  - a number of different heuristics have been developed
  - and deep results about the structure and relationships of these heuristics have been proved

Delete relaxed Heuristic

- If we remove all delete effects of all actions we get a new planning problem: the delete relaxed planning problem.
- Given a state S and goal G, consider the cost of a optimal delete relaxed plan for achieving G from S.
- This cost is a lower bound on the true optimal cost of moving from S to G—it is an admissible heuristic.
- This heuristic is called h^+

$$h^+(S) = \text{ the cost of an optimal delete relaxed plan for achieving } G \text{ from } S.$$

h^+

- $h^+ +$ is a very informative heuristic
- Computing $h^+(S)$ for an arbitrary state S is still NP-Hard!
- But it can be approximated well using MaxSat!
- This was done by Zhang and Bacchus and embedded in an A* search engine for computing optimal plans.
  1. Perform A* search for a plan.
  2. At each new state S approximate $h^+(S)$ using a MaxSat solver.
  3. Use the value returned by MaxSat to place S state on the OPEN list.

Computing Heuristics with MaxSat

- This worked surprisingly well (equaled the state-of-the-art at the time).
- However, although more work is needed to actually **advance** the state-of-the-art.
- In the meantime other heuristics have been developed (e.g., those based on solving linear-programs).
- On the other hand MaxSat solvers have made significant improvements....

MaxSat Encoding for Computing a Relaxed Plan

In delete relaxed planning, no action need be executed more than once.
And once a state fact becomes true it remains true (monotonic facts)

- For each state fact p define $sup(p)$ to be the set of actions that add p

$$sup(p) = \{a | a \wedge p \in add(a)\}$$

- For a state S define $poss\_acts(S)$ to be the set of actions executable in S

$$poss\_acts(S) = \{a | pre(a) \subseteq S\}$$

MaxSat Encoding for Computing a Relaxed Plan

- A propositional variable a_i for each action $a_i \in A$.

  a_i true means that the relaxed plan includes action a_i.
- For each goal $g \in G$ with $g \notin S$ we have the hard clause

$$\bigvee_{a \in sup(g)} a$$

  Some action in the plan must add g.
- For each action a and for each $p \in pre(a) \wedge p \notin S$ we have the hard clause

$$a \rightarrow \bigvee_{\{a' | a' \in sup(p)\}} a'$$

  If a is in the plan then all of its preconditions must be achieved (either by the initial state or by another action).
- For each action a we have soft clauses

$$(\neg a, aCost(a))$$

  If a is included in the plan we incur its cost.

Computing a Relaxed Plan with MaxSat

- This encoding is actually an approximate encoding.
- Motivation is to allow fast MaxSat solving so that heuristic computations can be done quickly.
- The problem with the encoding is that it admits cyclic plans.
  - Plans where a cycle of actions support each other's preconditions, but the preconditions are not achieved by any actions executable from the initial state.

Computing a Relaxed Plan with MaxSat

- The technique of constraint generation can be used.
- The original MaxSat problem does not include all of the constraints needed to solve the problem.
- When we obtain a solution, we check the solution to see if it is valid.
  - If it is valid, we have an optimal relaxed plan, and its cost is exactly $h^+(S)$.
  - If it is invalid we can compute a new clause that refines the MaxSat encoding, and resolve the MaxSat problem for a better solution.
- Each solution from the MaxSat encoding provides an improved **Lower Bound** on $h^+(S)$—so can be used as a heuristic value.
- If we continue until we get a valid relaxed plan we know that we have computed $h^+(S)$ exactly.

Computing a Relaxed Plan with MaxSat

The new clause to generate is simple.

1. Let A be the actions in the returned (invalid) solution.
2. Let $E = S$
3. Find an action $a \in A$ with $pre(a) \subseteq E$ and $add(a) \not\subseteq E$
4. $E = a(E)$
5. Repeat 3-4 until no more actions can be found.
6. Let A' be the set of actions executable E, that add something new to E.
7. The new clause to add to the MaxSat encoding is

$$\bigvee_{a \in A'} a$$

Computing a Relaxed Plan with MaxSat

- At least one action in E must be included in any plan from S to G.
  - A' is an action landmark.
- The process must eventually terminate.
- On termination we obtain a valid optimal relaxed plan.

Computing a Relaxed Plan with MaxSat

- This application illustrates that there are many ways to use MaxSat.
- One need not set up the entire problem as a MaxSat instance.
- Instead MaxSat can be used as a component in a more complex algorithm for solving the problem.

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-based, core-based, hybrids, ...
  - 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 — all you need to start looking further into MaxSat
- Survey of currently most relavant solving algorithms
  - core-based solvers
  - SAT-IP hybrids based on the implicit hitting set approach
  - branch-and-bound (still dominating approach to specific problem types)
- Overview of recent application domain of MaxSat (somewhat biased)
  - ideas for how to encode different problems as MaxSat
  - understanding some of the benefits of using MaxSat

Further Topics

In addition to what we covered today:
MaxSat is an active area of research, with recent work on

- preprocessing [Argelich, Li, and Manyà, 2008a] [Belov, Morgado, and Marques-Silva, 2013] [Berg, Saikko, and Järvisalo, 2015b] [Berg, Saikko, and Järvisalo, 2015a]
  - *How to simplify* MaxSat *instances to make the easier for solver(s)?*
- Parallel MaxSat solving [Martins, Manquinho, and Lynce, 2012] [Martins, Manquinho, and Lynce, 2015]
  - *How employ computing clusters to speed-up* MaxSat *solving?*
- Variants and generalization
  - MinSAT [Li, Zhu, Manyà, and Simon, 2012] [Argelich, Li, Manyà, and Zhu, 2013] [Ignatiev, Morgado, Planes, and Marques-Silva, 2013b] [Li and Manyà, 2015]
  - Quantified MaxSat [Ignatiev, Janota, and Marques-Silva, 2013a]

Further Topics

- instance decomposition/partitioning

  [Martins, Manquinho, and Lynce, 2013]

  [Neves, Martins, Janota, Lynce, and Manquinho, 2015]

- modelling high-level constraints [Argelich, Cabiscol, Lynce, and Manyà, 2012]

  [Zhu, Li, Manyà, and Argelich, 2012]

  [Heras, Morgado, and Marques-Silva, 2015]

- understanding problem/core structure

  [Li, Manyà, Mohamedou, and Planes, 2009]

  [Bacchus and Narodytska, 2014]

- Lower/upper bounds

  [Li, Manyà, and Planes, 2006]

  [Lin, Su, and Li, 2008]

  [Li, Manyà, Mohamedou, and Planes, 2010]

  [Li, Manyà, Mohamedou, and Planes, 2010]

  [Heras, Morgado, and Marques-Silva, 2012]

- symmetries [Marques-Silva, Lynce, and Manquinho, 2008]
- ...

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 v 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

Further Reading and Links

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, 2013]

MaxSat Evaluation `http://maxsat.ia.udl.cat`

Overview articles:
[Argelich, Li, Manyà, and Planes, 2008b]
[Argelich, Li, Manyà, and Planes, 2011]

Thank you for attending!

Bibliography I

Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), 2008.

Mario Alviano, Carmine Dodaro, and Francesco Ricca. A maxsat algorithm using cardinality constraints of bounded size. In Yang and Wooldridge [2015], pages 2677–2683. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-379.html.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. Solving (weighted) partial MaxSat through satisfiability testing. In Kullmann [2009], pages 427–440. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_39. URL http://dx.doi.org/10.1007/978-3-642-02777-2_39.

Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. SAT-based MaxSat algorithms. *Artif. Intell.*, 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. ISBN 978-1-61499-319-3. doi: 10.3233/978-1-61499-320-9-51. URL http://dx.doi.org/10.3233/978-1-61499-320-9-51.

Carlos Ansótegui, Frédéric Didier, and Joel Gabàs. Exploiting the structure of unsatisfiable cores in MaxSat. In Yang and Wooldridge [2015], pages 283–289. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-046.html.

Josep Argelich, Chu Min Li, and Felip Manyà. A preprocessor for max-sat solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, volume 4996 of *Lecture Notes in Computer Science*, pages 15–20. Springer, 2008a. ISBN 978-3-540-79718-0. doi: 10.1007/978-3-540-79719-7_2. URL http://dx.doi.org/10.1007/978-3-540-79719-7_2.

Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. The first and second max-sat evaluations. *JSAT*, 4(2-4):251–278, 2008b. URL http://jsat.ewi.tudelft.nl/content/volume4/JSAT4_13_Argelich.pdf.

Josep Argelich, Daniel Le Berre, Inês Lynce, João P. 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. doi: 10.4204/EPTCS.29.2. URL http://dx.doi.org/10.4204/EPTCS.29.2.

Bibliography II

Josep Argelich, Chu Min Li, Felip Manyà, and Jordi Planes. Experimenting with the instances of the MaxSat evaluation. In Cèsar Fernández, Hector Geffner, and Felip Manyà, editors, *Artificial Intelligence Research and Development - Proceedings of the 14th International Conference of the Catalan Association for Artificial Intelligence, Lleida, Catalonia, Spain, October 26-28, 2011*, volume 232 of *Frontiers in Artificial Intelligence and Applications*, pages 31–40. IOS Press, 2011. ISBN 978-1-60750-841-0. doi: 10.3233/978-1-60750-842-7-31. URL http://dx.doi.org/10.3233/978-1-60750-842-7-31.

Josep Argelich, Alba Cabiscol, Inês Lynce, and Felip Manyà. Efficient encodings from CSP into sat, and from maxcsp into MaxSat. *Multiple-Valued Logic and Soft Computing*, 19(1-3):3–23, 2012. URL http://www.oldcitypublishing.com/MVLSC/MVLSCabstracts/MVLSC18.5-6abstracts/MVLSCv18n5-6p445-456Zou.html.

Josep Argelich, Chu Min Li, Felip Manyà, and Zhu Zhu. Minsat versus MaxSat for optimization problems. In Schulte [2013], pages 133–142. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0_13. URL http://dx.doi.org/10.1007/978-3-642-40627-0_13.

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. ISBN 978-3-319-09283-6. doi: 10.1007/978-3-319-09284-3_2. URL http://dx.doi.org/10.1007/978-3-319-09284-3_2.

Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.

Anton Belov, António Morgado, and João Marques-Silva. Sat-based preprocessing for MaxSat. In McMillan et al. [2013], pages 96–111. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5_7. URL http://dx.doi.org/10.1007/978-3-642-45221-5_7.

Amir Ben-Dor, Ron Shamir, and Zohar Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6 (3/4):281–297, 1999.

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. ISBN 978-0-7695-5109-8. doi: 10.1109/ICDMW.2013.99. URL http://dx.doi.org/10.1109/ICDMW.2013.99.

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. ISBN 978-1-4799-6572-4. doi: 10.1109/ICTAI.2014.57. URL http://dx.doi.org/10.1109/ICTAI.2014.57.

Jeremias Berg and Matti Järvisalo. Cost-optimal constrained correlation clustering via weighted partial maximum satisfiability. *Artificial Intelligence*, 2016.

Jeremias Berg, Matti Järvisalo, and Brandon Malone. Learning optimal bounded treewidth bayesian networks via maximum satisfiability. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Proceedings*, pages 86–95. JMLR.org, 2014. URL http://jmlr.org/proceedings/papers/v33/berg14.html.

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, 2015a. ISBN 978-1-5090-0163-7. doi: 10.1109/ICTAI.2015.120. URL http://dx.doi.org/10.1109/ICTAI.2015.120.

Jeremias Berg, Paul Saikko, and Matti Järvisalo. Improving the effectiveness of sat-based preprocessing for MaxSat. In Yang and Wooldridge [2015], pages 239–245. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-040.html.

David Bergman, André A. Ciré, Willem Jan van Hoeve, and Tallys H. Yunes. Bdd-based heuristics for binary optimization. *J. Heuristics*, 20(2):211–234, 2014. doi: 10.1007/s10732-014-9238-1. URL http://dx.doi.org/10.1007/s10732-014-9238-1.

Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. In *Proc. ICDM*, pages 51–60. IEEE, 2011.

Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Antti Ukkonen. Chromatic correlation clustering. In *Proc. KDD*, pages 1321–1329. ACM, 2012.

Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for max-sat. *Artif. Intell.*, 171(8-9):606–618, 2007. doi: 10.1016/j.artint.2007.03.001. URL http://dx.doi.org/10.1016/j.artint.2007.03.001.

Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Tao Jiang. Correlation clustering and consensus clustering. In *Proc. ISAAC*, volume 3827 of *Lecture Notes in Computer Science*, pages 226–235. Springer, 2005. ISBN 3-540-30935-7.

Bibliography IV

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*, 2014. AAAI Press. ISBN 978-1-57735-661-5. URL http://www.aaai.org/Library/AAAI/aaai14contents.php.

Kerstin Bunte, Matti Järvisalo, Jeremias Berg, Petri Myllymäki, Jaakko Peltonen, and Samuel Kaski. Optimal neighborhood preserving visualization by maximum satisfiability. In Brodley and Stone [2014], pages 1694–1700. ISBN 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8242.

Nicolò Cesa-Bianchi, Claudio Gentile, Fabio Vitale, and Giovanni Zappella. A correlation clustering approach to link classification in signed networks. In *Proc. COLT*, volume 23 of *JMLR Procedings*, pages 34.1–34.20. JMLR.org, 2012.

Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. Clustering with qualitative information. *J. Comput. Syst. Sci.*, 71(3):360–383, 2005.

Yibin Chen, Sean Safarpour, Andreas G. Veneris, and João P. 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. ISBN 978-1-60558-522-2. doi: 10.1145/1531542.1531621. URL http://doi.acm.org/10.1145/1531542.1531621.

Yibin Chen, Sean Safarpour, João Marques-Silva, and Andreas G. Veneris. Automated design debugging with maximum satisfiability. *IEEE Trans. on CAD 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.

Ian Davidson and S. S. Ravi. Intractability and clustering with constraints. In *Proc. ICML*, pages 201–208. ACM, 2007.

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_19. URL http://dx.doi.org/10.1007/978-3-642-23786-7_19.

Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Schulte [2013], pages 247–262. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0_21. URL http://dx.doi.org/10.1007/978-3-642-40627-0_21.

Bibliography V

Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MaxSat. In Järvisalo and Gelder [2013], pages 166–181. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_13. URL http://dx.doi.org/10.1007/978-3-642-39071-5_13.

Jessica Davies, Jeremy Cho, and Fahiem Bacchus. Using learnt clauses in MaxSat. In David Cohen, editor, *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*, pages 176–190. Springer, 2010. ISBN 978-3-642-15395-2. doi: 10.1007/978-3-642-15396-9_17. URL http://dx.doi.org/10.1007/978-3-642-15396-9_17.

Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003. doi: 10.1145/636865.636866. URL http://doi.acm.org/10.1145/636865.636866.

Erik D. Demaine, Dotan Emanuel, Amos Fiat, and Nicole Immorlica. Correlation clustering in general weighted graphs. *Theor. Comput. Sci.*, 361(2-3):172–187, 2006.

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. ISBN 978-1-61499-418-3. doi: 10.3233/978-1-61499-419-0-303. URL http://dx.doi.org/10.3233/978-1-61499-419-0-303.

Vladimir Filkov and Steven Skiena. Heterogeneous data integration with the consensus clustering formalism. In *Proc. DILS*, volume 2994 of *Lecture Notes in Computer Science*, pages 110–123. Springer, 2004a. ISBN 3-540-21300-7.

Vladimir Filkov and Steven Skiena. Integrating microarray data by consensus clustering. *Int J Artif Intell T*, 13(4):863–880, 2004b.

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. doi: 10.1007/11814948_25. URL http://dx.doi.org/10.1007/11814948_25.

Jurgen Van Gael and Xiaojin Zhu. Correlation clustering for crosslingual link detection. In *Proc. IJCAI*, pages 1744–1749, 2007.

Bibliography VI

Raffaele Giancarlo and Filippo Utro. Speeding up the consensus clustering methodology for microarray data analysis. *Algorithms for Molecular Biology*, 6:1, 2011.

João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In Milano [2012], pages 941–956. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7_67. URL http://dx.doi.org/10.1007/978-3-642-33558-7_67.

Federico Heras, António Morgado, and João Marques-Silva. Lower bounds and upper bounds for MaxSat. In Youssef Hamadi and Marc Schoenauer, editors, *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, volume 7219 of *Lecture Notes in Computer Science*, pages 402–407. Springer, 2012. ISBN 978-3-642-34412-1. doi: 10.1007/978-3-642-34413-8_35. URL http://dx.doi.org/10.1007/978-3-642-34413-8_35.

Federico Heras, António Morgado, and Joao Marques-Silva. MaxSat-based encodings for group MaxSat. *AI Commun.*, 28(2): 195–214, 2015. doi: 10.3233/AIC-140636. URL http://dx.doi.org/10.3233/AIC-140636.

Antti Hyttinen, Frederick Eberhardt, and Matti Järvisalo. Constraint-based causal discovery: Conflict resolution with answer set programming. In Nevin L. Zhang and Jin Tian, editors, *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 340–349. AUAI Press, 2014. ISBN 978-0-9749039-1-0. URL https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2469&proceeding_id=30.

Alexey Ignatiev, Mikolás Janota, and João Marques-Silva. Quantified maximum satisfiability: - A core-guided approach. In Järvisalo and Gelder [2013], pages 250–266. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_19. URL http://dx.doi.org/10.1007/978-3-642-39071-5_19.

Alexey Ignatiev, António Morgado, Jordi Planes, and João Marques-Silva. Maximal falsifiability - definitions, algorithms, and applications. In McMillan et al. [2013], pages 439–456. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5_30. URL http://dx.doi.org/10.1007/978-3-642-45221-5_30.

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. ISBN 978-1-4503-2756-5. doi: 10.1145/2568225.2568306. URL http://doi.acm.org/10.1145/2568225.2568306.

Bibliography VII

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*, 2013. Springer. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5. URL http://dx.doi.org/10.1007/978-3-642-39071-5.

M. Jose and R. Majumdar. Cause clue clauses: error localization using maximum satisfiability. In *Proc. PLDI*, pages 437–446. ACM, 2011.

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*, 2009. Springer. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2. URL http://dx.doi.org/10.1007/978-3-642-02777-2.

Javier Larrosa and Federico Heras. Resolution in max-sat and its relation to local consistency in weighted csps. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 193–198. Professional Book Center, 2005. ISBN 0938075934. URL http://dblp.uni-trier.de/db/conf/ijcai/ijcai2005.html#LarrosaH05.

Chu Min Li and Felip Manyà. MaxSat, hard and soft constraints. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 613–631. IOS Press, 2009. ISBN 978-1-58603-929-5. 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 Felip Manyà. An exact inference scheme for minsat. In Yang and Wooldridge [2015], pages 1959–1965. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-278.html.

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. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1611.

Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for max-sat. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*, pages 86–91. AAAI Press, 2006. URL http://www.aaai.org/Library/AAAI/2006/aaai06-014.php.

Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Exploiting cycle structures in max-sat. In Kullmann [2009], pages 467–480. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_43. URL http://dx.doi.org/10.1007/978-3-642-02777-2_43.

Bibliography VIII

Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in maxsat. *Constraints*, 15(4):456–484, 2010. doi: 10.1007/s10601-010-9097-9. URL http://dx.doi.org/10.1007/s10601-010-9097-9.

Chu Min Li, Zhu Zhu, Felip Manyà, and Laurent Simon. Optimizing with minimum satisfiability. *Artif. Intell.*, 190:32–44, 2012. doi: 10.1016/j.artint.2012.05.004. URL http://dx.doi.org/10.1016/j.artint.2012.05.004.

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. ISBN 978-3-319-19083-9. doi: 10.1007/978-3-319-19084-6_26. URL http://dx.doi.org/10.1007/978-3-319-19084-6_26.

Han Lin, Kaile Su, and Chu Min Li. Within-problem learning for efficient lower bound computation in max-sat solving. In Dieter Fox and Carla P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 351–356. AAAI Press, 2008. ISBN 978-1-57735-368-3. URL http://www.aaai.org/Library/AAAI/2008/aaai08-055.php.

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

Vasco M. Manquinho, João P. Marques Silva, and Jordi Planes. Algorithms for weighted boolean optimization. In Kullmann [2009], pages 495–508. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_45. URL http://dx.doi.org/10.1007/978-3-642-02777-2_45.

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, Inês Lynce, and Vasco M. Manquinho. Symmetry breaking for maximum satisfiability. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 15th International Conference, LPAR 2008, Doha, Qatar, November 22-27, 2008. Proceedings*, volume 5330 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008. ISBN 978-3-540-89438-4. doi: 10.1007/978-3-540-89439-1_1. URL http://dx.doi.org/10.1007/978-3-540-89439-1_1.

João Marques-Silva, Mikolás Janota, Alexey Ignatiev, and António Morgado. Efficient model based diagnosis with maximum satisfiability. In Yang and Wooldridge [2015], pages 1966–1972. ISBN 978-1-57735-738-4. URL http://ijcai.org/papers15/Abstracts/IJCAI15-279.html.

Bibliography IX

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

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Community-based partitioning for MaxSat solving. In Järvisalo and Gelder [2013], pages 182–191. ISBN 978-3-642-39070-8. doi: 10.1007/978-3-642-39071-5_14. URL http://dx.doi.org/10.1007/978-3-642-39071-5_14.

Ruben Martins, Saurabh Joshi, Vasco M. Manquinho, and Inês Lynce. Incremental cardinality constraints for MaxSat. In O'Sullivan [2014], pages 531–548. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_39. URL http://dx.doi.org/10.1007/978-3-319-10428-7_39.

Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Deterministic parallel MaxSat solving. *International Journal on Artificial Intelligence Tools*, 24(3), 2015. doi: 10.1142/s0218213015500050. URL http://dx.doi.org/10.1142/s0218213015500050.

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*, 2013. Springer. ISBN 978-3-642-45220-8. doi: 10.1007/978-3-642-45221-5. URL http://dx.doi.org/10.1007/978-3-642-45221-5.

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*, 2012. Springer. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7. URL http://dx.doi.org/10.1007/978-3-642-33558-7.

Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013. doi: 10.1287/opre.1120.1139. URL http://dx.doi.org/10.1287/opre.1120.1139.

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

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, 2012. ISBN 978-3-642-39610-6. doi: 10.1007/978-3-642-39611-3_13. URL http://dx.doi.org/10.1007/978-3-642-39611-3_13.

Bibliography X

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.

António Morgado, Carmine Dodaro, and Joao Marques-Silva. Core-guided MaxSat with soft cardinality constraints. In O'Sullivan [2014], pages 564–573. ISBN 978-3-319-10427-0. doi: 10.1007/978-3-319-10428-7_41. URL http://dx.doi.org/10.1007/978-3-319-10428-7_41.

Nina Narodytska and Fahiem Bacchus. Maximum satisfiability using core-guided MaxSat resolution. In Brodley and Stone [2014], pages 2717–2723. ISBN 978-1-57735-661-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8513.

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_20. URL http://dx.doi.org/10.1007/978-3-319-24318-4_20.

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*, 2014. Springer. 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.

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. URL http://www.aaai.org/Library/AAAI/2002/aaai02-102.php.

Knot Pipatsrisawat, Akop Palyan, Mark Chavira, Arthur Choi, and Adnan Darwiche. Solving weighted max-sat problems in a reduced search space: A performance analysis. *JSAT*, 4(2-4):191–217, 2008. URL http://jsat.ewi.tudelft.nl/content/volume4/JSAT4_10_Pipatsrisawat.pdf.

Nathan Robinson, Charles Gretton, Duc Nghia Pham, and Abdul Sattar. Partial weighted maxsat for optimal planning. In Byoung-Tak Zhang and Mehmet A. Orgun, editors, *PRICAI 2010: Trends in Artificial Intelligence, 11th Pacific Rim International Conference on Artificial Intelligence, Daegu, Korea, August 30-September 2, 2010. Proceedings*, volume 6230 of *Lecture Notes in Computer Science*, pages 231–243. Springer, 2010. ISBN 978-3-642-15245-0. doi: 10.1007/978-3-642-15246-7_23. URL http://dx.doi.org/10.1007/978-3-642-15246-7_23.

Bibliography XI

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. ISBN 978-3-319-18007-6. doi: 10.1007/978-3-319-18008-3_24. URL http://dx.doi.org/10.1007/978-3-319-18008-3_24.

Paul Saikko, Johannes Peter Wallner, and Matti Järvisalo. Implicit hitting set algorithms for reasoning beyond NP. In Jim Delgrande and Frank Wolter, editors, *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, pages ???–??? AAAI Press, 2016.

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*, 2013. Springer. ISBN 978-3-642-40626-3. doi: 10.1007/978-3-642-40627-0. URL http://dx.doi.org/10.1007/978-3-642-40627-0.

Kiri Wagstaff and Claire Cardie. Clustering with instance-level constraints. In *Proc. ICML*, pages 1103–1110. Morgan Kaufmann, 2000.

Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained K-means clustering with background knowledge. In *Proc. ICML*, pages 577–584. Morgan Kaufmann, 2001.

Johannes Peter Wallner, Andreas Niskanen, and Matti Järvisalo. Complexity results and algorithms for extension enforcement in abstract argumentation. In Dale Schuurmans and Michael Wellman, editors, *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016)*, pages ???–??? AAAI Press, 2016.

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*, 2015. AAAI Press. ISBN 978-1-57735-738-4.

Zhiwen Yu, Hau-San Wong, and Hong-Qiang Wang. Graph-based consensus clustering for class discovery from gene expression data. *Bioinformatics*, 23(21):2888–2896, 2007.

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. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5190.

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. ISBN 978-0-9835678-1-3. URL http://dl.acm.org/citation.cfm?id=2157667.

Bibliography XII

Zhu Zhu, Chu Min Li, Felip Manyà, and Josep Argelich. A new encoding from minsat into MaxSat. In Milano [2012], pages 455–463. ISBN 978-3-642-33557-0. doi: 10.1007/978-3-642-33558-7_34. URL http://dx.doi.org/10.1007/978-3-642-33558-7_34.