Algorithms for Maximum Satisfiability  
with Applications to AI

Fahiem Bacchus¹  Matti Järvisalo²

¹University of Toronto  
Canada

²University 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

Motivation and Basic Concepts
  Exact Optimization
  Benefits of MaxSat
  MaxSat: Basic Definitions
  MaxSat Solvers: Input Format, Evaluations, and Availability

Algorithms for MaxSat Solving
  Branch and Bound
  MaxSat by Integer Programming
  SAT-Based MaxSat Solving
    Iterative Search
    Core-based Approaches
  SAT-IP Hybrid Algorithms for MaxSat
  Iterative Use of SAT Solvers for MaxSat

Modelling and Applications
  Representing High-Level Soft Constraints in MaxSat
  MaxSat-based Cost-optimal Correlation Clustering
  Heuristics for Planning using MaxSat
Success of SAT

The Boolean satisfiability (SAT) Problem
Input: A propositional logic formula $F$.
Task: Is $F$ satisfiable?
Success of SAT

The Boolean satisfiability (SAT) Problem
Input: A propositional logic formula $F$.
Task: Is $F$ satisfiable?

SAT is a Great Success Story
Not merely a central problem in theory:
Remarkable improvements since mid 90s in SAT solvers:
practical decision procedures for SAT
- Find solutions if they exist
- Prove non-existence of solutions
SAT Solvers

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

Results of the SAT competition/race winners on the SAT 2009 application benchmarks, 20mn timeout

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]
maximum quartet consistency [Chen, Safarpour, Marques-Silva, and Veneris, 2010]
software package management [Morgado and Marques-Silva, 2010]
Max-Clique [Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
fault localization [Ignatiev, Janota, and Marques-Silva, 2014]
restoring CSP consistency [Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
reasoning over bionetworks [Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
MCS enumeration [Lynce and Marques-Silva, 2011]
heuristics for cost-optimal planning [Guerra and Lynce, 2012]
optimal covering arrays [Morgado, Liffiton, and Marques-Silva, 2012]
correlation clustering [Zhang and Bacchus, 2012]
treewidth computation [Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
Bayesian network structure learning [Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
causal discovery [Berg and Järvisalo, 2014]
visualisation [Berg, Järvisalo, and Malone, 2014]
model-based diagnosis [Hyttinen, Eberhardt, and Järvisalo, 2014]
cutting planes for IPs [Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
argumentation dynamics [Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
...
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]
[Morgado and Marques-Silva, 2010]
maximum quartet consistency
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
software package management
[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 and Järvisalo, 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]
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 (\( \lor \)) of literals. e.g. \((x \lor y \lor \neg z)\)
- **Truth assignment** \( \tau \): 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 \( \tau \).
MaxSat: Basic Definitions

MaxSat
INPUT: a set of clauses $F$. \hspace{1cm} (a CNF formula)
TASK: find $\tau$ s.t. $\sum_{C \in F} \tau(C)$ is maximized.

Find truth assignment that satisfies a maximum number of clauses

This is the standard definition, much studied in Theoretical Computer Science.
- Often inconvenient for modeling practical problems.
Central Generalizations of MaxSat

Weighted MaxSat

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

Partial MaxSat

- Some clauses are deemed hard—infinite weights
  - Any solution has to satisfy the hard clauses
    $\leadsto$ 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 if path visits this square.
MaxSat: Example

- **Boolean variables**: one for each unblocked grid square \( \{S, G, a, b, \ldots, u\} \): true if 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

```
<table>
<thead>
<tr>
<th>n</th>
<th>o</th>
<th>p</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>l</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>f</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>b</td>
<td>g</td>
<td>m</td>
</tr>
</tbody>
</table>
```
MaxSat: Example

Constraint 1:

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>o</td>
<td>p</td>
<td>q</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>G</td>
</tr>
<tr>
<td>c</td>
<td>d</td>
<td>e</td>
<td>l</td>
<td>r</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>b</td>
<td>g</td>
<td>m</td>
<td>u</td>
</tr>
</tbody>
</table>
MaxSat: Example

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

- For S: \(a + b = 1\)
  - In CNF: \((a \lor b), \neg a \lor \neg b\)
MaxSat: Example

Constraint 1:

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

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

\(\text{disallow pairwise}\)
MaxSat: Example

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

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

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

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

Encoding Cardinality Constraints in CNF

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

Properties of the encoding

- Every solution to the hard clauses is a path from S to G that does not pass a blocked square.
- Such a path will falsify one negative soft clause for every square it passes through.
  - **orange path**: assign 14 variables in \{S, a, c, h, . . . , t, r, G\} to true
- MaxSat solutions: paths that pass through a minimum number of squares (i.e., is shortest).
  - **green path**: assign 8 variables in \{S, b, g, f, . . . , 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 $\text{FP}^{\text{NP}}$–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 $\text{FP}^{\text{NP}}$–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 $\text{FP}^\text{NP}$–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 $\text{FP}^\text{NP}$–complete problems include TSP
- A SAT solver acts as the NP oracle most often in practice

MaxSat is hard to approximate $\text{APX}$–complete

$\text{APX}$: class of NP optimization problems that
- admit a constant-factor approximation algorithm, \textit{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 \(\neg x_3\)
- \(\emptyset\): 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
  - \((-x_3 \lor x_1 \lor \neg x_{45})\), 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
... 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, \textit{no command line parameters necessary}
- Input: CNF formula, in the \textit{standard} DIMACS WCNF file format
- Output: provably optimal solution, or UNSATISFIABLE
  - Complete solvers

Internally rely especially on CDCL SAT solvers for proving unsatisfiability of subsets of clauses

```
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
```
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__
- MSCG http://sat.inesc-id.pt/-aign/soft/
- WPM3 http://web.udl.es/usuarios/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  
- 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/usuarios/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 $\Phi$ has hard and soft clauses:
  
  $c_i \in \Phi: \text{cost}(c_i)$ is weight
  
  $\text{cost}(c_i) < \infty$ soft clause $\quad \text{cost}(c_i) = \infty$ hard clause

- Truth assignment $\pi$
  
  $\text{cost}(\pi)$ is sum of the weights of clauses falsified by $\pi$.

- $\pi$ satisfies all hard clauses: it is a solution.

- $\text{cost}(\pi)$ is minimum over all solutions: it is an optimal solution.

- MaxSat algorithms often cast as minimization problem: find solution with minimum $\text{cost}$ (optimal).
Branch and Bound
Branch and Bound

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

$$\Phi = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots$$

Ignoring clause costs, $$\kappa = \{(x) \land (\neg x)\}$$ is inconsistent.
Lower Bounds

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

\( \Phi = \ldots \land (x, 2)\ldots \land (\neg x, 3)\ldots \)

Ignoring clause costs, \( \kappa = \{(x) \land (\neg x)\} \) is inconsistent.

Let \( \kappa' = \{ (\square, 2) \land (\neg x, 1) \} \).

Then \( \kappa' \) is MaxSat-equivalent to \( \kappa \): the cost of each truth assignment is preserved. (\( \square \) is empty clause)
Common LB technique in MaxSat solvers: look for inconsistencies that force some soft clause to be falsified.

\[ \Phi = \ldots \land (x, 2) \ldots \land (\neg x, 3) \ldots \]

Ignoring clause costs, \( \kappa = \{(x) \land (\neg x)\} \) is inconsistent.

Let \( \kappa' = \{(\Box, 2) \land (\neg x, 1)\} \).

Then \( \kappa' \) is MaxSat-equivalent to \( \kappa \): the cost of each truth assignment is preserved. (\( \Box \) is empty clause)

Let \( \Phi' = \Phi - \kappa \cup \kappa' \).

Then \( \Phi' \) is MaxSat-equivalent to \( \Phi \), and the cost of \( \Box \) has been incremented by 2.

Cost of \( \Box \) must be incurred: it is an LB.
Lower Bounds

1. Detect an inconsistent subset $\kappa$ (aka core) of the current formula
   - e.g. $\kappa = \{ (x, 2) \land (\neg x, 3) \}$

2. Apply sound transformation to the clauses in $\kappa$ that result in an increment to the cost of the empty clause $\square$
   - e.g. $\kappa$ replaced by $\kappa' = \{ (\square, 2) \land (\neg x, 1) \}$
     - 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 \[ (x \lor a_1 \lor \ldots \lor a_s \lor w_1), (\neg x \lor b_1 \lor \ldots \lor b_t, w_2) \] =

\[ (a_1 \lor \ldots \lor a_s \lor b_1 \lor \ldots \lor b_t, \min(w_1, w_2)) \quad \text{Regular Resolvent} \]
\[ (x \lor a_1 \lor \ldots \lor a_s, w_1 - \min(w_1, w_2)) \quad \text{Cost Reduced Input} \]
\[ (\neg x \lor b_1 \lor \ldots \lor b_t, w_2 - \min(w_1, w_2)) \quad \text{One will vanish} \]
\[ (x \lor a_1 \lor \ldots \lor a_s \lor \neg(b_1 \lor \ldots \lor b_t), \min(w_1, w_2)) \quad \text{Compensation Clauses} \]
\[ (\neg x \lor \neg(a_1 \lor \ldots \lor a_s) \lor b_1 \lor \ldots \lor b_t, \min(w_1, w_2)) \quad \text{must be converted to Clauses} \]

[Larrosa and Heras, 2005; Bonet, Levy, and Manyà, 2007]
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 \((\square, \text{Opt})\) with weight \(\text{Opt}\) 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 \((x_1 \lor x_2 \lor \cdots \lor x_k)\) we add a new variable \(b\):

\[
(b \lor x_1 \lor x_2 \lor \cdots \lor x_k)
\]

\(b\) 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 $b_i$.

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

- 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

  $$ \text{minimize} \sum_{i} b_i \times \text{cost}(c_i) $$
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 $\Phi$ is SAT
   2.1 Try to SAT solve $\Phi$.
   2.2 If SAT STOP found optimal solution
   2.3 Else relax $\Phi$ so that more soft clauses can be falsified.
      ▶ Minimum relaxation $\Rightarrow$ optimal solution when $\Phi$ 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 $\Phi$
2. Add blocking variable $b_i$ to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If $\text{SAT}(\Phi \cup \text{CNF}(\sum b_i \leq k))$ return $k$
5. Else $k = k + 1$ and repeat 4.
Iterative SAT solving (Linear Search)

1. Input MaxSat CNF $\Phi$
2. Add blocking variable $b_i$ to every soft clause $c_i \in \Phi$
3. Set $k = 0$.
4. If $\text{SAT}(\Phi \cup \text{CNF}(\sum b_i \leq k))$ 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: 
\( (\neg a), \ldots, (\neg t) \iff (\neg a \lor b_a), \ldots, (\neg u \lor b_u) \).

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 \( (a \lor b) \) (must exit \( S \)) falsified \( \Rightarrow \) UNSAT.
   - Hard clause \( (k \lor q \lor r) \) (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 \( (\neg e \lor b_e) \) 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 \( (a \lor b) \) 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 $\pi$ of the hard clauses.
   3.2 Solve with $k = (\# \text{ of clauses falsified by } \pi) - 1$
   3.3 If SAT found better assignment. Reset $k$ and repeat 2.
   3.4 If UNSAT last assignment $\pi$ 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 $\Phi$, a core of $\Phi$ is a subset of $\Phi$ that is itself unsatisfiable.

Cores for MaxSat
A subset of soft clauses of $\Phi$ that together with the hard clauses of $\Phi$ 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
- \((\kappa, SAT?) = \text{sat}(\Phi)\)
  Sat solve \(\Phi\). Return Boolean SAT or UNSAT status: \(SAT?\)
  If UNSAT, return a core \(\kappa\) (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 $\kappa$ is a MaxSat core, then at least one if the soft clauses in it must be removed: no truth assignment satisfies every clause in $\kappa$ along with all of the hard clauses.
- Typically cores are *much* smaller than the set of all soft clauses.
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
1. Input MaxSat CNF $\Phi$
2. $k = 0; BV = \{\}.$
3. $(\kappa, SAT?) = \text{sat}(\Phi)$
4. If $SAT?$ return $k.$
5. $k = k + 1$
6. Update $\Phi$:
   - 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.
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.
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.
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 
  \((b_a + b_b \leq 1) \land (b_c + b_g \leq 1)\).
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
  $(\neg a \lor b_a), (\neg b \lor b_b)$ 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

$$(a \land (b \lor c \lor g)) \lor (b \land (c \lor g))$$
Overlapping Cores

- For these cores the constraint
  \[(b_a + b_b \leq 1) \land (b_b + b_c + b_g \leq 1)\] is too strong: E.g.
  \(b \land c\) is a solution of
  \[(a \land (b \lor c \lor g)) \lor (b \land (c \lor g))\]
  but is not a solution of the two cardinality constraints
  \[(b_a + b_b \leq 1) \land (b_b + b_c + b_g \leq 1)\]

- 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. $k = 0, \kappa = \{(-a), (-b)\}$

2. Update these clauses to $\{(a, b_a), (-b, b_b)\}$.

3. Add $CNF(b_a + b_a \leq 1)$

4. $k = 1, \kappa = \{(-b, b_b), (-c), (-g)\}$

5. Update these clauses to $\{(a, b_b, b_{b}^1), (-c, b_c), (-g, b_g)\}$

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 $\leq 1$ so can be encoded more efficiently.

3. Fu-Malik contains some key ideas but no longer state of the art.
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 \textit{at-most} \( \leq \) cardinality constraint over the soft clauses in a cover. Disjoint so this works.
- Keep an \textit{at-least} \( \geq \) constraint over the clauses in a core.
WPM2

Cover 2

Core 2
\[ d_1 + d_2 + d_3 \geq 1 \]

Core 2
\[ d_3 + e_1 \geq 1 \]

Core 4
\[ e_1 + e_2 + e_3 + e_4 \geq 1 \]

\[ d_1 + d_2 + d_3 + e_1 + e_2 + e_3 + e_4 \leq 3 \]
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)
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.
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.
Processing a new core in EVA

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

Processing a new core in EVA

1. \( \text{Core} = \{c_1, c_2, c_3, c_4, c_5\} \)

New core

2. \( b_1 \equiv c_1 \quad b_2 \equiv c_2 \quad b_3 \equiv c_3 \)

new variables \( b_i \) equivalent to \( c_i \)

\( b_4 \equiv c_4 \quad b_5 \equiv c_5 \)
Processing a new core in EVA

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

2. \begin{align*}
   b_1 & \equiv c_1 \\
   b_2 & \equiv c_2 \\
   b_3 & \equiv c_3 \\
   b_4 & \equiv c_4 \\
   b_5 & \equiv c_5
\end{align*}

3. Remove softs \{c_1, c_2, c_3, c_4, c_5\}

New core

new variables $b_i$ equivalent to $c_i$
Processing a new core in EVA

1. Core = \{c_1, c_2, c_3, c_4, c_5\}
2. \(b_1 \equiv c_1\) \(b_2 \equiv c_2\) \(b_3 \equiv c_3\) \(b_4 \equiv c_4\) \(b_5 \equiv c_5\)
3. Remove softs \(\{c_1, c_2, c_3, c_4, c_5\}\)
4. \((-b_1 \lor -b_2 \lor -b_3 \lor -b_4 \lor -b_5)\) must falsify one of the \(c_i\)
Processing a new core in EVA

1. \[ \text{Core} = \{c_1, c_2, c_3, c_4, c_5\} \]

2. \[ \begin{align*} b_1 & \equiv c_1 \\ b_2 & \equiv c_2 \\ b_3 & \equiv c_3 \\ b_4 & \equiv c_4 \\ b_5 & \equiv c_5 \end{align*} \]

3. **Remove softs** \( \{c_1, c_2, c_3, c_4, c_5\} \)

4. \[ (\neg b_1 \lor \neg b_2 \lor \neg b_3 \lor \neg b_4 \lor \neg b_5) \]

5. \[ \begin{align*} d_1 & \equiv b_2 \land b_3 \land b_4 \land b_5 \\ d_2 & \equiv b_3 \land b_4 \land b_5 \\ d_3 & \equiv b_4 \land b_5 \\ d_4 & \equiv b_5 \end{align*} \]

New core

new variables \( b_i \) equivalent to \( c_i \)

must falsify one of the \( c_i \)

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. Core = \{c_1, c_2, c_3, c_4, c_5\}
2. \begin{align*}
b_1 & \equiv c_1 & b_2 & \equiv c_2 & b_3 & \equiv c_3 \\
b_4 & \equiv c_4 & b_5 & \equiv c_5
\end{align*}
3. Remove softs \{c_1, c_2, c_3, c_4, c_5\}
4. \((\neg b_1 \lor \neg b_2 \lor \neg b_3 \lor \neg b_4 \lor \neg b_5)\) must falsify one of the \(c_i\)
5. \begin{align*}d_1 & \equiv b_2 \land b_3 \land b_4 \land b_5 \\
d_2 & \equiv b_3 \land b_4 \land b_5 \\
d_3 & \equiv b_4 \land b_5 \\
d_4 & \equiv b_5
\end{align*}
6. \begin{align*}(b_1 \lor d_1, 1) \\
(b_2 \lor d_2, 1) \\
(b_3 \lor d_3, 1) \\
(b_4 \lor d_4, 1)
\end{align*}

New core
new variables \(b_i\) equivalent to \(c_i\)
new variables \(d_i\)
\(d_i\) indicates \(c_{i+1} \ldots c_5\) satisfied
new softs
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 \( (b_i \lor d_i, 1) \) satisfied by \( b_i \).
  3. Consider \( (b_3, d_3, 1) \)
      \[ d_3 \equiv b_4 \land b_5 \text{ is TRUE} \]
      so this soft clause is satisfied.

No cost is incurred in the new formula. (new formula is relaxed)
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. $d_2 \equiv b_3 \land b_4 \land b_5$ is FALSE
     $(b_2 \vee d_2, 1)$ is FALSIFIED.
  4. $d_3 \equiv b_4 \land b_5$ 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.
\( d_i \) variables capture a disjunction of soft clauses (c.f. extended resolution)

If a future core involves \((\neg b_i \lor \neg d_i, 1)\) we get a new variable \(x_i \equiv (\neg b_1 \lor \neg d_i)\).

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
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. $w_{\text{min}}$ is minimum weight in $K$.
3. We split each clause $(c, w) \in K$ into two clauses
   
   (1) $(c, w_{\text{min}})$  
   (2) $(c, w - w_{\text{min}})$.
4. Keep all clauses (2) $(c, w - w_{\text{min}})$ as soft clauses
   (discard zero weight clauses)
5. We let $K$ be all clauses (1) $(c, w_{\text{min}})$
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 $S$ of sets of elements, 
A set $H$ is a hitting set of $S$ if $H \cap S \neq \emptyset$ for all $S \in S$.
A hitting set $H$ is optimal if no $H' \subset \bigcup S$ with $|H'| < |H|$ is a hitting set of $S$. 

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 to $F$ such that satisfies exactly the clauses $F_n H$.
Hitting Sets and UNSAT Cores

Hitting Sets
Given a collection $S$ of sets of elements,
A set $H$ is a hitting set of $S$ if $H \cap S \neq \emptyset$ for all $S \in S$.

A hitting set $H$ is optimal if no $H' \subset \bigcup S$ with $|H'| < |H|$ is a hitting set of $S$.

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 $\tau$ to $F$ such that $\tau$ 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

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

...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

\[
\begin{align*}
\min & \quad \sum_{C \in \mathcal{U} \mathcal{K}} c(C) \cdot b_C \\
\text{subject to} & \quad \sum_{C \in K} b_C \geq 1 \quad \forall K \in \mathcal{K}
\end{align*}
\]

▶ $b_C = 1$ 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 $c : F_s \rightarrow \mathbb{R}^+$

SAT solver
$F_h \land (F_s \setminus hs)$

**UNSAT core extraction**

Min-cost Hitting Set

IP solver
\[
\min \sum_{c \in \mathcal{K}} c(C) \cdot b_c \\
\sum_{c \in K} b_c \geq 1 \quad \forall K \in \mathcal{K}
\]

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

1. Initialize

   $F_h, F_s$
   $hs := \emptyset$
   $\mathcal{K} := \emptyset$

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

   $c$

   $F_h \wedge (F_s \setminus hs)$

   SAT solver

   UNSAT core extraction

   $hs$ of $\mathcal{K}$

   sat

   min $\sum_{C \in \mathcal{K}} c(C) \cdot b_C$
   $\sum_{C \in \mathcal{K}} b_C \geq 1 \ \forall K \in \mathcal{K}$

   IP solver

   Optimal solution found

   unsat

   Min-cost Hitting Set
Solving MaxSat by SAT and Hitting Set Computations

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

2. UNSAT core

\[
F_h, F_s \\
h s := \emptyset \\
K := \emptyset
\]

SAT solver

\[
F_h \land (F_s \setminus hs)
\]

Min-cost Hitting Set

\[
c
\]

IP solver

\[
\min \sum_{c \in K} c(C) \cdot b_C \\
\sum_{c \in K} b_C \geq 1 \quad \forall K \in K
\]

Optimal solution found
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

$F_h, F_s$
$hs := \emptyset$
$\mathcal{K} := \emptyset$
$\mathcal{K} := \mathcal{K} \cup \{K\}$

**SAT solver**
$F_h \land (F_s \setminus hs)$

**UNSAT core extraction**

**Min-cost Hitting Set**

**IP solver**
$\min \sum_{C \in \mathcal{K}} c(C) \cdot b_C$
$\sum_{C \in K} b_C \geq 1 \ \forall K \in \mathcal{K}$

unsat

$hs$ of $\mathcal{K}$

Optimal solution found
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 $\mathcal{K}$

$F_h, F_s$
$hs := \emptyset$
$\mathcal{K} := \emptyset$
$\mathcal{K} := \mathcal{K} \cup \{K\}$

**SAT solver**

$F_h \land (F_s \setminus hs)$

**UNSAT core extraction**

unsat

**Min-cost Hitting Set**

$hs$ of $\mathcal{K}$

**IP solver**

\[
\min \sum_{C \in \mathcal{K}} c(C) \cdot b_C
\]

\[
\sum_{C \in K} b_C \geq 1 \forall K \in \mathcal{K}
\]

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

5. UNSAT core

\[
F_h, F_s \hspace{1cm} h_s := \emptyset \hspace{1cm} K := \emptyset
\]

\[
K := K \cup \{K\}
\]

\[
\text{IP solver} \hspace{1cm} \min \sum_{C \in K} c(C) \cdot b_C \hspace{1cm} \sum_{C \in K} b_C \geq 1 \ \forall K \in K
\]

Optimal solution found
Solving MaxSat by SAT and Hitting Set Computations

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

iterate until "sat"

\[ F_h, F_s \]
\[ hs := \emptyset \]
\[ K := \emptyset \]

\[ F_h \wedge (F_s \setminus hs) \]

**SAT solver**

**UNSAT core extraction**

\[ \mathcal{K} := \mathcal{K} \cup \{K\} \]

\[ \text{UNSAT core extraction} \]

\[ hs \text{ of } \mathcal{K} \]

**Min-cost Hitting Set**

\[ \text{IP solver} \]

\[ \min \sum_{c \in \mathcal{K}} c(C) \cdot b_C \]
\[ \sum_{c \in K} b_C \geq 1 \forall K \in \mathcal{K} \]

\[ \text{Optimal solution found} \]
Solving MaxSat by SAT and Hitting Set Computations

**Input:**
- hard clauses $F_h$, soft clauses $F_s$, weight function $c : F_s \mapsto \mathbb{R}^+$
- iterate until “sat”

**SAT solver**
- $F_h \land (F_s \setminus hs)$

**UNSAT core extraction**
- $F_h, F_s$
- $hs := \emptyset$
- $\mathcal{K} := \emptyset$

**Min-cost Hitting Set**
- $\mathcal{K} := \mathcal{K} \cup \{K\}$
- $c$

**IP solver**
- $\min \sum_{C \in \mathcal{K}} c(C) \cdot b_C$
- $\sum_{C \in \mathcal{K}} b_C \geq 1 \ \forall K \in \mathcal{K}$

**Optimal solution found**
- $hs$ of $\mathcal{K}$

- $sat$

- $unsat$
Intuition: After optimally hitting all cores of $F_h \land F_s$ by $hs$: any solution to $F_h \land (F_s \setminus hs)$ is guaranteed to be optimal.

iterate until “sat”

$F_h, F_s$
$hs := \emptyset$
$K := \emptyset$

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

unsat

$IP$ solver

$\min \sum_{C \in \mathcal{K}} c(C) \cdot b_C$
$\sum_{C \in \mathcal{K}} b_C \geq 1 \ \forall K \in \mathcal{K}$

Optimal solution found
MaxSat by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \]
\[ C_4 = \neg x_1 \]
\[ C_7 = x_2 \lor x_4 \]
\[ C_{10} = \neg x_7 \lor x_5 \]
\[ C_2 = \neg x_6 \lor x_2 \]
\[ C_5 = \neg x_6 \lor x_8 \]
\[ C_8 = \neg x_4 \lor x_5 \]
\[ C_{11} = \neg x_5 \lor x_3 \]
\[ C_3 = \neg x_2 \lor x_1 \]
\[ C_6 = x_6 \lor \neg x_8 \]
\[ C_9 = x_7 \lor x_5 \]
\[ C_{12} = \neg x_3 \]
MaxSat by SAT and Hitting Set Computation: Example

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

\[ \mathcal{K} := \emptyset \]
MaxSat by SAT and Hitting Set Computation: Example

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

\[ K := \emptyset \]

- SAT solve \( F_h \land (F_s \setminus \emptyset) \)
MaxSat by SAT and Hitting Set Computation: Example

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

\[\cal{K} := \emptyset\]

- SAT solve \( F_h \land (F_s \setminus \emptyset) \leadsto \) UNSAT core \( K = \{C_1, C_2, C_3, C_4\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Update \( \mathcal{K} := \mathcal{K} \cup \{ K \} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

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

\( \leadsto hs = \{C_1\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

\[ \text{SAT solve } F_h \land (F_s \setminus \{C_1\}) \]
MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve \( F_h \land (F_s \setminus \{C_1\}) \leadsto \) UNSAT core \( K = \{C_9, C_{10}, C_{11}, C_{12}\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Update \( \mathcal{K} := \mathcal{K} \cup \{K\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
  \[ \sim h_s = \{C_1, C_9\} \]
MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve \( F_h \land (F_5 \setminus \{ C_1, C_9 \}) \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve \( F_h \land (F_s \setminus \{C_1, C_9\}) \)
\[ \leadsto \text{UNSAT core} \ K = \{C_3, C_4, C_7, C_8, C_{11}, C_{12}\} \]
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Update \( \mathcal{K} := \mathcal{K} \cup \{K\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over \( K \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- Solve minimum-cost hitting set problem over \( \mathcal{K} \)
  \( \leadsto hs = \{C_4, C_9\} \)
MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve \( F_h \land (F_s \setminus \{C_4, C_9\}) \)
MaxSat by SAT and Hitting Set Computation: Example

\[ C_1 = x_6 \lor x_2 \quad C_2 = \neg x_6 \lor x_2 \quad C_3 = \neg x_2 \lor x_1 \]

\[ C_4 = \neg x_1 \quad C_5 = \neg x_6 \lor x_8 \quad C_6 = x_6 \lor \neg x_8 \]

\[ C_7 = x_2 \lor x_4 \quad C_8 = \neg x_4 \lor x_5 \quad C_9 = x_7 \lor x_5 \]

\[ C_{10} = \neg x_7 \lor x_5 \quad C_{11} = \neg x_5 \lor x_3 \quad C_{12} = \neg x_3 \]

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

\[ \text{▶ SAT solve } F_h \land (F_s \setminus \{C_4, C_9\}) \leadsto \text{SATISFIABLE.} \]
MaxSat by SAT and Hitting Set Computation: Example

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

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

- SAT solve \( F_h \land (F_s \setminus \{C_4, C_9\}) \) \( \Rightarrow \) SATISFIABLE.
  Optimal cost: 2 (cost of \( hs \)).
Optimizations in Solvers

Solvers implementing the implicit hitting 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

- 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 \( a_i \)
  - \( \sim \) replace \( C_i \) with \((C_i \lor a_i)\) 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 \( a_i \)s
  - 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
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 $C$ with associated weight $W_C$.

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

Softening $\text{CNF}(C)$ 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 $W_C$. 
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 $C$ with associated weight $W_C$.

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

Softening $\text{CNF}(C)$ as Weighted Partial MaxSat:

- **Hard clauses**: $\bigwedge_{i=1}^{m} (C_i \lor a)$, where $a$ is a fresh Boolean variable
- **Soft clause**: $(a)$ with weight $W_C$.

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]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
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
[Lyne and Marques-Silva, 2011]
reasoning over bionetworks
[Guerra and Lynce, 2012]
MCS enumeration
[Morgado, Lifftion, 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  
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]  
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]  
[Park, 2002]

design debugging  
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]  
[Morgado and Marques-Silva, 2010]  
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]

maximum quartet consistency  
[Morgado and Marques-Silva, 2010]  
[Ignatiev, Janota, and Marques-Silva, 2014]

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]  
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]  
[Lynce and Marques-Silva, 2011]  
[Guerra and Lynce, 2012]  
[Morgado, Liffton, and Marques-Silva, 2012]  
[Zhang and Bacchus, 2012]  
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]  
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]  
[Berg and Järvisalo, 2014]  
[Berg, Järvisalo, and Malone, 2014]  
[Hyttinen, Eberhardt, and Järvisalo, 2014]  
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]  
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]  
[Saikkko, Malone, and Järvisalo, 2015]  
[Wallner, Niskanen, and Järvisalo, 2016]

fault localization

restoring CSP consistency

reasoning over bionetworks

MCS enumeration

heuristics for cost-optimal planning

optimal covering arrays

correlation clustering

treewidth computation

Bayesian network structure learning

causal discovery

visualization

model-based diagnosis

cutting planes for IPs

argumentation dynamics

...
MaxSat Applications

probabilistic inference
[Park, 2002]
design debugging
[Chen, Safarpour, Veneris, and Marques-Silva, 2009]
[Chen, Safarpour, Marques-Silva, and Veneris, 2010]
[Argelich, Berre, Lynce, Marques-Silva, and Rapicault, 2010]
[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]
Max-Clique
[Ignatiev, Janota, and Marques-Silva, 2014]
fault localization
[Li and Quan, 2010; Fang, Li, Qiao, Feng, and Xu, 2014; Li, Jiang, and Xu, 2015]
restoring CSP consistency
[Zhu, Weissenbacher, and Malik, 2011; Jose and Majumdar, 2011]
reasoning over bionetworks
[Lyne and Marques-Silva, 2011]
MCS enumeration
[Guerra and Lynce, 2012]
heuristics for cost-optimal planning
[Morgan, Liffiton, and Marques-Silva, 2012]
optimal covering arrays
[Zhang and Bacchus, 2012]
correlation clustering
[Ansótegui, Izquierdo, Manyà, and Torres-Jiménez, 2013b]
treewidth computation
[Berg and Järvisalo, 2013; Berg and Järvisalo, 2016]
Bayesian network structure learning
[Berg and Järvisalo, 2014]
causal discovery
[Berg, Järvisalo, and Malone, 2014]
visualization
[Hyttinen, Eberhardt, and Järvisalo, 2014]
model-based diagnosis
[Bunte, Järvisalo, Berg, Myllymäki, Peltonen, and Kaski, 2014]
cutting planes for IPs
[Marques-Silva, Janota, Ignatiev, and Morgado, 2015]
argumentation dynamics
[Saikko, Malone, and Järvisalo, 2015]
...
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* [Bansal, Blum, and Chawla, 2004]
  - 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

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

  [Filkov and Skiena, 2004b,a; Giancarlo and Utro, 2011; Yu, Wong, and Wang, 2007]
Cost-Optimal Correlation Clustering

<table>
<thead>
<tr>
<th>V</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1$</td>
<td>0.5</td>
<td>1</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>$v_2$</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>...</td>
</tr>
<tr>
<td>$v_3$</td>
<td>0.7</td>
<td>1</td>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>$v_4$</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>$v_5$</td>
<td>6</td>
<td>0</td>
<td>10</td>
<td>...</td>
</tr>
</tbody>
</table>

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

$\Rightarrow$ MAXSAT: encoding + solving

DATA

SIMILARITY MATRIX

SOLUTION CLUSTERING

**INPUT:** a similarity matrix $W$,

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

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

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

where the indicator function $I[b] = 1$ 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**

Minimize \( \sum_{-\infty < W(i,j) < 0, i < j} (x_{ij} \cdot |W(i,j)|) - \sum_{\infty > W(i,j) > 0, i < j} (x_{ij} \cdot W(i,j)) \)

where \( x_{ij} + x_{jk} \leq 1 + x_{ik} \) for all distinct \( i, j, k \)

\( x_{ij} = 1 \) for all \( W(i, j) = \infty \)

\( x_{ij} = 0 \) for all \( W(i, j) = -\infty \)

\( x_{ij} \in \{0, 1\} \) for all \( i, j \)

**Transitivity-based encoding**

\( O(n^2) \) variables and \( O(n^3) \) 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
- $O(n^2)$ variables and $O(n^3)$ clauses.
- Same indicator variables: $x_{ij} = 1$ iff $cl(v_i) = cl(v_j)$

**Hard clauses**

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

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

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

**Encoding $W(i,j) = -\infty$**: $(\neg x_{ij})$
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 \( x_{ij} + x_{jk} \leq 1 + x_{ik} \):

- \((x_{ij} \land x_{jk}) \rightarrow x_{ik}\)
  as clause: \( (\neg x_{ij} \lor \neg x_{jk} \lor x_{ik}) \)

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

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

**Soft clauses**

Encoding the objective function:

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

\[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]
Example

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

- Hard clauses:
  $$\{ (\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12}) \}$$

- 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:
  \[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]

- 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:
  \[ \{(\neg x_{12} \lor \neg x_{23} \lor x_{13}), (\neg x_{12} \lor \neg x_{13} \lor x_{23}), (\neg x_{23} \lor \neg x_{13} \lor x_{12})\} \]

- Soft clauses:
  \[ \{(x_{12}; 1), (x_{13}; 1), (\neg x_{23}; 1)\} \]

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 \( b_1 \ldots b_{\log |D|} \)
  - Interpret an assignment to \( b_1 \ldots b_{\log |D|} \) as the bit-representation of a value in \( D \).

Does not always pay off 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 $b^k_i$ for $k = 1..\log N$.
▶ $S_{ij} = 1$ iff points $i$ and $j$ are co-clustered
▶ Auxiliary: $EQ^k_{ij} = 1$ iff $b^k_i = b^k_j$

Hard clauses

▶ Semantics of $EQ^k_{ij}$: $EQ^k_{ij} \iff (b^k_i \iff b^k_j)$
▶ Semantics of $S_{ij}$: $S_{ij} \iff (EQ^1_{ij} \land \cdots \land EQ^\log N_{ij})$
▶ Encoding $W(i,j) = \infty$: $(S_{ij})$
  Encoding $W(i,j) = -\infty$: $(\neg S_{ij})$

Soft clauses

▶ For $W(i,j) \in (0, \infty)$: $(S_{ij})$ with weight $W(i,j)$
▶ For $W(i,j) \in (-\infty, 0)$: $(\neg S_{ij})$ 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:**
  
  \[
  S_{12} \leftrightarrow (EQ_{12}^1 \land EQ_{12}^2) \\
  S_{13} \leftrightarrow (EQ_{13}^1 \land EQ_{13}^2) \\
  S_{23} \leftrightarrow (EQ_{23}^1 \land EQ_{23}^2)
  \]

  \[
  EQ_{12}^1 \leftrightarrow (b_1^1 \leftrightarrow b_1^2) \\
  EQ_{12}^2 \leftrightarrow (b_1^2 \leftrightarrow b_2^2) \\
  EQ_{13}^1 \leftrightarrow (b_1^1 \leftrightarrow b_1^3) \\
  EQ_{13}^2 \leftrightarrow (b_1^2 \leftrightarrow b_3^3) \\
  EQ_{23}^1 \leftrightarrow (b_2^1 \leftrightarrow b_1^3) \\
  EQ_{23}^2 \leftrightarrow (b_2^2 \leftrightarrow b_3^2)
  \]

- **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:

\[ S_{12} \leftrightarrow (EQ_{12}^1 \land EQ_{12}^2) \]

\[ S_{13} \leftrightarrow (EQ_{13}^1 \land EQ_{13}^2) \]

\[ S_{23} \leftrightarrow (EQ_{23}^1 \land EQ_{23}^2) \]

- Soft clauses:

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

\[ EQ_{12}^1 \leftrightarrow (b_1^1 \leftrightarrow b_2^1) \]

\[ EQ_{12}^2 \leftrightarrow (b_2^1 \leftrightarrow b_2^2) \]

\[ EQ_{13}^1 \leftrightarrow (b_1^1 \leftrightarrow b_3^1) \]

\[ EQ_{13}^2 \leftrightarrow (b_1^2 \leftrightarrow b_3^2) \]

\[ EQ_{23}^1 \leftrightarrow (b_2^1 \leftrightarrow b_3^1) \]

\[ EQ_{23}^2 \leftrightarrow (b_1^2 \leftrightarrow b_3^2) \]
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)
     - \textit{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 \( G \subseteq P \)
     - \textit{no Closed world for G}: \( p \notin G \not\leftrightarrow \neg p \)
  5. Actions \( A \)
  6. Action cost function \( aCost: a \in A \) incurs cost \( aCost(a) > 0 \).
Actions

- Each action \( a \in A \) is \( \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle \)
- Preconditions \( \text{pre}(a) \); add effects \( \text{add}(a) \), and \( \text{del}(a) \) delete effects.
- Actions map states \( S \) to new states \( a(S) \):
  1. If \( \text{pre}(a) \not\subseteq S \) we can’t apply \( a \) to \( S \) (\( a \) is not executable in \( S \)).
  2. If \( a \) is executable in \( S \) then
     \[
     a(S) = S - \text{del}(a) \cup \text{add}(a).
     \]
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 $\Pi$ is the sum of $aCost(a_i)$ for all $a_i \in \Pi$.
- Given a planning problem we want to find a plan $\Pi$ 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 an 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....
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 \( p \)

\[
sup(p) = \{ a | a \land p \in add(a) \}
\]

- 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 $p \in pre(a) \land p \notin S$ 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 relevant 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
  - How to simplify MaxSat instances to make the easier for solver(s)?
  - Parallel MaxSat solving
    - How employ computing clusters to speed-up MaxSat solving?
  - Variants and generalization
    - MinSAT
    - Quantified MaxSat

[Argelich, Li, and Manyà, 2008a]
[Belov, Morgado, and Marques-Silva, 2013]
[Berg, Saikko, and Järvisalo, 2015b]
[Berg, Saikko, and Järvisalo, 2015a]
[Martins, Manquinho, and Lynce, 2012]
[Martins, Manquinho, and Lynce, 2015]
[Li, Zhu, Manyà, and Simon, 2012]
[Argelich, Li, Manyà, and Zhu, 2013]
[Ignatiev, Morgado, Planes, and Marques-Silva, 2013b]
[Li and Manyà, 2015]
[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 vs propagation
  - Underlying core-structure of encodings
  - The “best” solvers for current benchmark sets may not be best for novel applications!
    - Requires trial-and-error & in-depth understanding of solvers and the problem domain
Further Reading and Links

Surveys

▶ Handbook chapter on MaxSat: [Li and Manyà, 2009]
▶ Surveys on MaxSat algorithms: [Ansótegui, Bonet, and Levy, 2013a]
[Argelich, Li, Manyà, and Planes, 2011]

MaxSat Evaluation

Overview articles: http://maxsat.ia.udl.cat
[Argelich, Li, Manyà, and Planes, 2008b]
[Argelich, Li, Manyà, and Planes, 2011]
Thank you for attending!
Bibliography


Bibliography III


Bibliography V


Bibliography VII


Bibliography VIII


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.


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.