SAT Solving and its Extensions

A Short Introduction of Maximum Satisfiability

Uwe Egly, Katalin Fazekas

Slides are mostly based on Chapter 24 of SAT Handbook [BacchusJärvisaloMartins-2021]

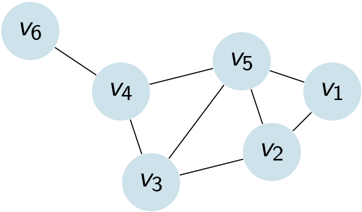
Institute of Logic and Computation
TU Wien, Austria

Motivation – Vertex Cover

- Many problems have an *objective function* beyond the Boolean constraints

Motivation – Vertex Cover

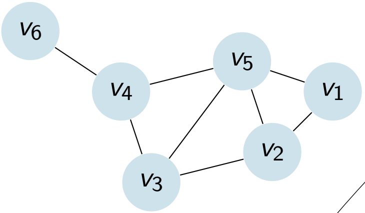
- Many problems have an *objective function* beyond the Boolean constraints



- Find a subset of the vertices that includes at least one endpoint of every edge of the graph

Motivation – Vertex Cover

- Many problems have an *objective function* beyond the Boolean constraints



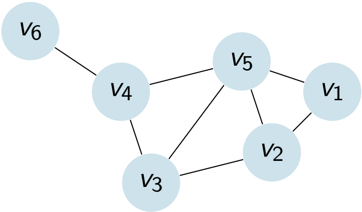
- Find a subset of the vertices that includes at least one endpoint of every edge of the graph

SAT Encoding:

1. Introduce a Boolean variable for each vertex v_i ($i \in [1..6]$)
   - v_i is true $\leftrightarrow$ vertex v_i is selected to be part of the cover
2. Add a clause $(v_i \vee v_j)$ for each $(v_i, v_j) \in E$ where $(i < j)$

Motivation – Vertex Cover

- Many problems have an *objective function* beyond the Boolean constraints



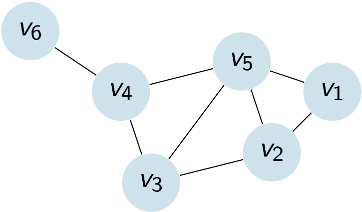
- Find a subset of the vertices that includes at least one endpoint of every edge of the graph
  - For example $\{v_6, v_5, v_4, v_3, v_2, v_1\}$

SAT Encoding:

1. Introduce a Boolean variable for each vertex v_i ($i \in [1..6]$)
   - v_i is true $\leftrightarrow$ vertex v_i is selected to be part of the cover
2. Add a clause $(v_i \vee v_j)$ for each $(v_i, v_j) \in E$ where $(i < j)$

Motivation – Vertex Cover

- Many problems have an *objective function* beyond the Boolean constraints



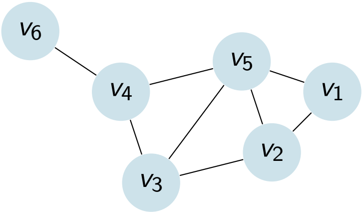
- Find a subset of the vertices that includes at least one endpoint of every edge of the graph
  - For example $\{v_6, v_5, v_4, v_3, v_2, v_1\}$
- Find the **smallest** subset:

SAT Encoding:

1. Introduce a Boolean variable for each vertex v_i ($i \in [1..6]$)
   - v_i is true $\leftrightarrow$ vertex v_i is selected to be part of the cover
2. Add a clause $(v_i \vee v_j)$ for each $(v_i, v_j) \in E$ where $(i < j)$
3. **Objective function:** Minimize $v_1 + v_2 + v_3 + v_4 + v_5 + v_6$

Motivation – Vertex Cover

- Many problems have an *objective function* beyond the Boolean constraints



- Find a subset of the vertices that includes at least one endpoint of every edge of the graph
  - For example $\{v_6, v_5, v_4, v_3, v_2, v_1\}$
- Find the **smallest** subset:
  - $\{v_5, v_4, v_2\}$

SAT Encoding:

1. Introduce a Boolean variable for each vertex v_i ($i \in [1..6]$)
   - v_i is true $\leftrightarrow$ vertex v_i is selected to be part of the cover
2. Add a clause $(v_i \vee v_j)$ for each $(v_i, v_j) \in E$ where $(i < j)$
3. **Objective function:** Minimize $v_1 + v_2 + v_3 + v_4 + v_5 + v_6$

MaxSAT Applications

- Verification and Security
  - design and debug circuits, error trace minimization, hardware-software partitioning, user-guided program analysis, malware detection ...
- Planning, Scheduling, Configuration
  - cost-optimal planning, shortest path, course timetabling, wedding seating arrangements, package upgradeability, vehicle configurations, ...
- AI and Data Analysis Problems
  - most probably explanations in Bayesian networks, interpretable classification rules, constrained correlation clustering, visualizations
- Combinatorial Problems
  - max-clique problem, Steiner tree problem, tree-width computation, ...
- Bioinformatics
  - haplotype inference problem, maximal similarities between RNA sequences, ...
- ...

Preliminaries

Maximum Satisfiability (MaxSAT)

MaxSAT:

- Boolean optimization problems expressed in CNF, partitioned into two parts:

$$F = \mathit{hard}(F) \cup \mathit{soft}(F)$$

  - Hard clauses: must be satisfied
  - Soft clauses, each with a weight: falsifying incurs a cost equal with the weight

Example

$$F = \underbrace{(x \vee y) \wedge (\neg x \vee r \vee z)}_{\text{hard clauses}} \wedge \underbrace{(\neg x)_2 \wedge (\neg y)_5}_{\text{soft clauses}}$$

Goal:

- Find a truth assignment that satisfies all the hard clauses and a maximum weight of soft clauses.

Maximum Satisfiability (MaxSAT) (cont.)

Definition (Feasible solution)

Given a MaxSAT formula F, a truth assignment to the variables of F, $vars(F)$, that satisfies $hard(F)$ is a **feasible solution**.

- The cost of a feasible solution τ of F is the sum of the weights of the soft clauses it falsifies:

$$cost(\tau, F) = \Sigma_{\{C \mid C \in soft(F) \wedge \tau \neg\models C\}} wt(C)$$

Maximum Satisfiability (MaxSAT) (cont.)

Definition (Feasible solution)

Given a MaxSAT formula F, a truth assignment to the variables of F, $vars(F)$, that satisfies $hard(F)$ is a **feasible solution**.

- The cost of a feasible solution τ of F is the sum of the weights of the soft clauses it falsifies:

$$cost(\tau, F) = \Sigma_{\{C \mid C \in soft(F) \wedge \tau \neg\models C\}} wt(C)$$

Optimal solution:

An **optimal solution** τ of F is a feasible solution with minimum cost: $cost(\tau, F) \leq cost(\tau', F)$ for every feasible solution τ' of F.

Maximum Satisfiability (MaxSAT) (cont.)

Definition (The MaxSAT Problem)

Given a MaxSAT formula F, find one of its optimal solutions.

- We assume that $hard(F)$ is satisfiable

Maximum Satisfiability (MaxSAT) (cont.)

Definition (The MaxSAT Problem)

Given a MaxSAT formula F, find one of its optimal solutions.

- We assume that $hard(F)$ is satisfiable

Definition (MaxSAT Core)

Given a MaxSAT formula F, any subset K of $soft(F)$ such that $K \cup hard(F)$ is unsatisfiable, is a **core** of F.

$\rightarrow$ We always have to satisfy $hard(F)$, so more useful to define cores relative to the hard clauses

$\rightarrow$ Every feasible solution of F falsifies at least one soft clause of K

Maximum Satisfiability (MaxSAT) (cont.)

Definition (The MaxSAT Problem)

Given a MaxSAT formula F, find one of its optimal solutions.

- We assume that $\mathit{hard}(F)$ is satisfiable

Definition (MaxSAT Core)

Given a MaxSAT formula F, any subset K of $\mathit{soft}(F)$ such that $K \cup \mathit{hard}(F)$ is unsatisfiable, is a **core** of F.

→ We always have to satisfy $\mathit{hard}(F)$, so more useful to define cores relative to the hard clauses

→ Every feasible solution of F falsifies at least one soft clause of K

$$F = (x \lor \neg y \lor v) \land (\neg y \lor z) \land \underbrace{(y \lor z)_{10} \land (\neg z)_5}_{\text{K}} \land (x \lor \neg v)_7$$

Some Variants of MaxSAT

MaxSAT

- all clauses are soft and and each one has weight 1
- satisfy maximum number of soft clauses

Partial MaxSAT

- there are hard clauses and soft clauses with weight 1
- satisfy maximum number of soft clauses while satisfy all hard clauses

Weighted MaxSAT

- all clauses are soft and have a positive weight associated with
- maximize sum of weights of satisfied soft clauses

Weighted Partial MaxSAT

- there are hard clauses and soft clauses with positive weights
- maximize sum of weights of satisfied soft clauses while satisfy all hard clauses

Incremental SAT-based MaxSAT Solving

How to solve MaxSAT?

- Use Branch-and-Bound methods
- Use an Integer Programming (IP) or Pseudo-Boolean (PB) solver

Incremental SAT-based MaxSAT Solving

How to solve MaxSAT?

- Use Branch-and-Bound methods
- Use an Integer Programming (IP) or Pseudo-Boolean (PB) solver
- Use a SAT solver:
  - To simplify the presentation, from now on we will assume that every soft clause has weight 1 (unless stated otherwise)

Incremental SAT-based MaxSAT Solving

How to solve MaxSAT?

- Use Branch-and-Bound methods
- Use an Integer Programming (IP) or Pseudo-Boolean (PB) solver
- Use a SAT solver:
  - To simplify the presentation, from now on we will assume that every soft clause has weight 1 (unless stated otherwise)
  - Solve a sequence of SAT queries where each query asks the following:

    Is there a solution of the hard clauses that falsifies at most k soft clauses?

Incremental SAT-based MaxSAT Solving

How to solve MaxSAT?

- Use Branch-and-Bound methods
- Use an Integer Programming (IP) or Pseudo-Boolean (PB) solver
- Use a SAT solver:
  - To simplify the presentation, from now on we will assume that every soft clause has weight 1 (unless stated otherwise)
  - Solve a sequence of SAT queries where each query asks the following:

    Is there a solution of the hard clauses that falsifies at most k soft clauses?

  - How to encode this question?

Incremental SAT-based MaxSAT Solving

How to solve MaxSAT?

- Use Branch-and-Bound methods
- Use an Integer Programming (IP) or Pseudo-Boolean (PB) solver
- Use a SAT solver:
  - To simplify the presentation, from now on we will assume that every soft clause has weight 1 (unless stated otherwise)
  - Solve a sequence of SAT queries where each query asks the following:

    Is there a solution of the hard clauses that falsifies at most k soft clauses?

  - How to encode this question?
  - How to exploit incremental SAT solvers?

Iterative Search Algorithms

Solving MaxSAT with Iterative Search

1. Linear search UNSAT $\rightarrow$ SAT (Lower Bound search)

   Can we find a solution that falsifies at most 0 of the soft clauses?
   If not, can we find a solution that falsifies at most 1 of the soft clauses?
   If not, can we find a solution that falsifies at most 2 of the soft clauses?
   ...

Solving MaxSAT with Iterative Search

1. Linear search UNSAT $\to$ SAT (Lower Bound search)

   Can we find a solution that falsifies at most 0 of the soft clauses?
   If not, can we find a solution that falsifies at most 1 of the soft clauses?
   If not, can we find a solution that falsifies at most 2 of the soft clauses?
   ...

2. Linear search SAT $\to$ UNSAT (Upper Bound search)

   Can we find a solution that satisfies at least 1 of the soft clauses?
   ($\sim$ falsifies at most all but one of the soft clauses?)
   If not, can we find a solution that satisfies at least 2 of the soft clauses?
   If not, can we find a solution that satisfies at least 3 of the soft clauses?
   ...

Solving MaxSAT with Iterative Search

Project idea

1. Linear search UNSAT → SAT (Lower Bound search)

   Can we find a solution that falsifies at most 0 of the soft clauses?
   If not, can we find a solution that falsifies at most 1 of the soft clauses?
   If not, can we find a solution that falsifies at most 2 of the soft clauses?
   ...

2. Linear search SAT → UNSAT (Upper Bound search) + incremental SAT solver

   Can we find a solution that satisfies at least 1 of the soft clauses?
   (∼ falsifies at most all but one of the soft clauses?)
   If not, can we find a solution that satisfies at least 2 of the soft clauses?
   If not, can we find a solution that satisfies at least 3 of the soft clauses?
   ...

3. Binary search until LB = UB - 1

Linear UNSAT $\rightarrow$ SAT search

Algorithm 1 UNSAT-SAT Search

F^b = bv_transform(F); $k = 0$;
(sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \text{CNF}(\Sigma \neg b_i \leq k))$
while not sat? **do**
    k $\leftarrow$ k $+1$;
    (sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \text{CNF}(\Sigma \neg b_i \leq k))$
return k

Linear UNSAT $\rightarrow$ SAT search

Algorithm 1 UNSAT-SAT Search

$F^b = \text{bv\_transform}(F)$; $k = 0$;
(sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \text{CNF}(\Sigma \neg b_i \leq k))$
while not sat? **do**
    k $\leftarrow$ k $+1$;
    (sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \text{CNF}(\Sigma \neg b_i \leq k))$
return k

- Not really effective without using unsat cores (see later)

Linear UNSAT $\rightarrow$ SAT search

→ Project MAXSAT naive

Algorithm 1 UNSAT-SAT Search

F^b = bv_transform(F); $k = 0$;
(sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \mathrm{CNF}(\Sigma \neg b_i \leq k))$
while not sat? **do**
    k $\leftarrow$ k $+1$;
    (sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F^b \cup \mathrm{CNF}(\Sigma \neg b_i \leq k))$
return k

- Not really effective without using unsat cores (see later)
- Constraints like $\mathrm{CNF}(\Sigma \neg b_i \leq k)$ will be part of the *card_layer* of the problem in the later algorithms

Linear SAT $\rightarrow$ UNSAT search

Algorithm 2 SAT-UNSAT Search [BerreParrain-2010]

F^b = bv_transform(F); card_layer = {}; Blits = $\{\neg b_i \mid (b_i) \in soft(F^b)\}$
bestModel = {}; cost = inf; sat? = *true*;
while sat? **do**
    (sat?,τ,κ) $\leftarrow$ `SAT`($F^b \cup$ card_layer))
    if sat? **then**
        bestModel $\leftarrow$ τ;
        cost $\leftarrow$ cost(τ);
        card_layer $\leftarrow$ $\{\text{CNF}(\Sigma_{b_i \in Blits} \neg b_i < cost)\}$;
    else
        return (bestModel$|_{vars(F)}$, cost)

Linear SAT $\rightarrow$ UNSAT search

Algorithm 2 SAT-UNSAT Search [BerreParrain-2010]

F^b = bv_transform(F); card_layer = {}; Blits = $\{\neg b_i \mid (b_i) \in \mathit{soft}(F^b)\}$
bestModel = {}; cost = inf; sat? = *true*;
while sat? **do**
  (sat?,τ,κ) $\leftarrow$ `SAT`($F^b \cup$ card_layer))
  if sat? **then**
    bestModel $\leftarrow \tau$;
    cost $\leftarrow$ cost(τ);
    card_layer $\leftarrow$ {CNF($\Sigma_{b_i \in Blits} \neg b_i < \mathit{cost}$)};
  else
    return (bestModel$|_{vars(F)}$, cost)

- Used in QMaxSAT [KoshimuraZFH-2012] and Pacose [BacchusJM-MaxSAT Evaluation 2018]
- Can be effective in certain cases + Can give approximate solutions

Binary Search

Combines SAT-UNSAT and UNSAT-SAT search:

1. $UB \leftarrow |Blits|$, $LB \leftarrow 0$
2. Try to find a solution with $k = \frac{UB+LB}{2}$ soft clauses
3. If satisfiable, update the upper bound to k
4. If unsatisfiable, update the lower bound to k
5. Repeat 2-4. until $UB == LB + 1$

Binary Search

Combines SAT-UNSAT and UNSAT-SAT search:

1. $UB \leftarrow |Blits|$, $LB \leftarrow 0$
2. Try to find a solution with $k = \frac{UB+LB}{2}$ soft clauses
3. If satisfiable, update the upper bound to k
4. If unsatisfiable, update the lower bound to k
5. Repeat 2-4. until $UB == LB + 1$

- Can be effective (if considers cores)

Binary Search

Combines SAT-UNSAT and UNSAT-SAT search:

1. $UB \leftarrow |Blits|$, $LB \leftarrow 0$
2. Try to find a solution with $k = \frac{UB+LB}{2}$ soft clauses
3. If satisfiable, update the upper bound to k
4. If unsatisfiable, update the lower bound to k
5. Repeat 2-4. until $UB == LB + 1$

- Can be effective (if considers cores)
- Potentially has less iterations than the other two iterative approaches

Core Guided Algorithms

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i$ was added in step $1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i \text{ was added in step } 1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

$$F = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y)_{10} \wedge (\neg z)_5$$

$$F^b = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y \vee \neg b_1) \wedge (\neg z \vee \neg b_2) \wedge (b_1)_{10} \wedge (b_2)_5$$

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i$ was added in step $1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

$$F = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y)_{10} \wedge (\neg z)_5$$
$$F^b = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y \vee \neg b_1) \wedge (\neg z \vee \neg b_2) \wedge (b_1)_{10} \wedge (b_2)_5$$

- unit soft clauses are usually not transformed (here we will in the examples)

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i$ was added in step $1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

$$F = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y)_{10} \wedge (\neg z)_5$$
$$F^b = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y \vee \neg b_1) \wedge (\neg z \vee \neg b_2) \wedge (b_1)_{10} \wedge (b_2)_5$$

- unit soft clauses are usually not transformed (here we will in the examples)
- F^b has only unit soft clauses ($\rightarrow$ use of assumptions, see next slide)

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i$ was added in step $1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

$$F = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y)_{10} \wedge (\neg z)_5$$

$$F^b = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y \vee \neg b_1) \wedge (\neg z \vee \neg b_2) \wedge (b_1)_{10} \wedge (b_2)_5$$

- unit soft clauses are usually not transformed (here we will in the examples)
- F^b has only unit soft clauses ($\rightarrow$ use of assumptions, see next slide)
- If $\tau^b \models F^b$ then $\tau^b|_{vars(F)} \models F$ and $cost(\tau^b, F^b) \geq cost(\tau^b|_{vars(F)}, F)$

Blocking Variable Transformation

Definition

Given a MaxSAT instance F, F^b is a new MaxSAT instance with:

1. $hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$
2. $soft(F^b) = \{(b_i) \mid b_i$ was added in step $1\}$
3. $wt((b_i)) = wt(c_i)$ (each new soft clause (b_i) gets the weight of the original $C_i \in soft(F)$).

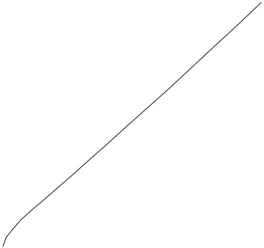
$$F = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y)_{10} \wedge (\neg z)_5$$

$$F^b = (x \vee \neg y \vee z) \wedge (\neg y \vee z) \wedge (z \vee y \vee \neg b_1) \wedge (\neg z \vee \neg b_2) \wedge (b_1)_{10} \wedge (b_2)_5$$

- unit soft clauses are usually not transformed (here we will in the examples)
- F^b has only unit soft clauses ($\rightarrow$ use of assumptions, see next slide)
- If $\tau^b \models F^b$ then $\tau^b\mid_{vars(F)} \models F$ and $cost(\tau^b, F^b) \geq cost(\tau^b\mid_{vars(F)}, F)$
- If $\tau \models F$, then there is an extension τ^b of it with $cost(\tau^b, F^b) = cost(\tau, F)$

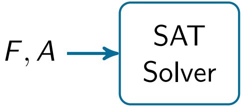
UNSAT Cores via Assumptions of Incremental SAT

- Modern MaxSAT algorithms rely on unsat cores



UNSAT Cores via Assumptions of Incremental SAT

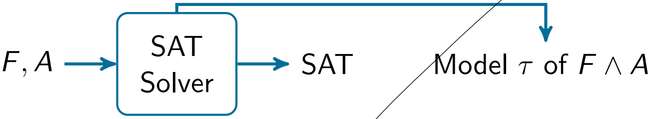
- Modern MaxSAT algorithms rely on unsat cores
- Assumption based incremental SAT solving can provide it:



1. Solve F subject to a set of assumption literals A (query `SAT`(F,assume=A))

UNSAT Cores via Assumptions of Incremental SAT

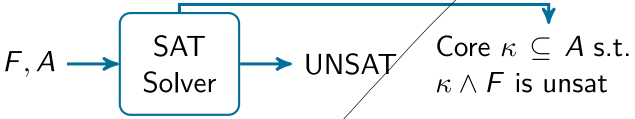
- Modern MaxSAT algorithms rely on unsat cores
- Assumption based incremental SAT solving can provide it:



1. Solve F subject to a set of assumption literals A (query $\texttt{SAT}(F,\text{assume}=A)$)
2. If answer is SAT, then the solution τ satisfies all hard clauses and all literals in A

UNSAT Cores via Assumptions of Incremental SAT

- Modern MaxSAT algorithms rely on unsat cores
- Assumption based incremental SAT solving can provide it:

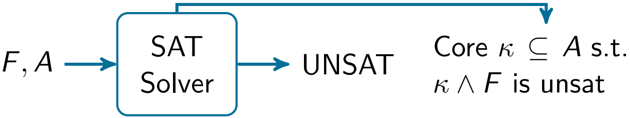


1. Solve F subject to a set of assumption literals A (query `SAT`(F,assume=A))
2. If answer is SAT, then the solution τ satisfies all hard clauses and all literals in A
3. If answer is UNSAT, the solver returns a set of literals $\kappa = \{l_1, l_2, \ldots, l_n\}$ such that $\kappa \subseteq A$ and $F \models \neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_n$

   $\rightarrow$ any model of F must falsify at least one of the literals of κ

UNSAT Cores via Assumptions of Incremental SAT

- Modern MaxSAT algorithms rely on unsat cores
- Assumption based incremental SAT solving can provide it:



1. Solve F subject to a set of assumption literals A (query SAT(F,assume=A))
2. If answer is SAT, then the solution τ satisfies all hard clauses and all literals in A
3. If answer is UNSAT, the solver returns a set of literals $\kappa = \{l_1, l_2, \ldots, l_n\}$ such that $\kappa \subseteq A$ and $F \models \neg l_1 \vee \neg l_2 \vee \ldots \vee \neg l_n$

   $\rightarrow$ any model of F must falsify at least one of the literals of κ

- Cores are not necessarily minimal

MaxSAT Cores via Assumptions of Incremental SAT

Reminder of **Blocking Variable Transformation:**

$$hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$$
$$soft(F^b) = \{(b_i) \mid b_i \text{ was added in step } 1\}$$

MaxSAT Cores via Assumptions of Incremental SAT

Reminder of **Blocking Variable Transformation:**

$$hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$$
$$soft(F^b) = \{(b_i) \mid b_i \text{ was added in step 1}\}$$

- Use the blocking literals as assumptions
  - If these are assumed to be true, their corresponding soft clauses must be satisfied

MaxSAT Cores via Assumptions of Incremental SAT

Reminder of **Blocking Variable Transformation:**

$$hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$$
$$soft(F^b) = \{(b_i) \mid b_i \text{ was added in step 1}\}$$

- Use the blocking literals as assumptions
  - If these are assumed to be true, their corresponding soft clauses must be satisfied
- Solve query $\texttt{SAT}(hard(F^b)$,assume$=\{b_1, b_2, \ldots, b_n\})$:

MaxSAT Cores via Assumptions of Incremental SAT

Reminder of **Blocking Variable Transformation:**

$$hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$$
$$soft(F^b) = \{(b_i) \mid b_i \text{ was added in step } 1\}$$

- Use the blocking literals as assumptions
  - If these are assumed to be true, their corresponding soft clauses must be satisfied
- Solve query $\texttt{SAT}(hard(F^b)$,assume$=\{b_1, b_2, \ldots, b_n\})$:
  - If answer is SAT, solution τ satisfies all hard clauses and makes every b_i true and so τ satisfies every soft clauses as well

MaxSAT Cores via Assumptions of Incremental SAT

Reminder of **Blocking Variable Transformation:**

$$hard(F^b) = hard(F) \cup \{(C_i \vee \neg b_i \mid C_i \in soft(F)\}$$
$$soft(F^b) = \{(b_i) \mid b_i \text{ was added in step } 1\}$$

- Use the blocking literals as assumptions
  - If these are assumed to be true, their corresponding soft clauses must be satisfied
- Solve query $\texttt{SAT}(hard(F^b), \text{assume}=\{b_1, b_2, \ldots, b_n\})$:
  - If answer is SAT, solution τ satisfies all hard clauses and makes every b_i true and so τ satisfies every soft clauses as well
  - If answer is UNSAT, the returned core κ is actually a MaxSAT core (subset of soft clauses s.t. $hard(F) \cup \kappa$ is unsatisfiable)

Cardinality Constraints

Is there a solution of the hard clauses that falsifies **at most** k soft clauses?

Cardinality Constraints

Is there a solution of the hard clauses that falsifies **at most** k soft clauses?

- Cardinality constraints must be encoded as CNF:

$$\Sigma_{i=1}^{n} b_i \bowtie k \text{ where } \bowtie \in \{\leq, =, \geq\}$$

  - At-Most-1 (AMO) constraints: $\Sigma_{i=1}^{n} b_i \leq 1$
  - General cardinality constraints: $\Sigma_{i=1}^{n} b_i \leq k$
  - (Pseudo-Boolean constraints: $\Sigma_{i=1}^{n} w_i b_i \leq k$ where $w_i, k \in \mathbb{Z}$)

Cardinality Constraints

Is there a solution of the hard clauses that falsifies **at most** k soft clauses?

- Cardinality constraints must be encoded as CNF:

$$\Sigma_{i=1}^{n} b_i \bowtie k \text{ where } \bowtie \in \{\leq, =, \geq\}$$

  - At-Most-1 (AMO) constraints: $\Sigma_{i=1}^{n} b_i \leq 1$
  - General cardinality constraints: $\Sigma_{i=1}^{n} b_i \leq k$
  - (Pseudo-Boolean constraints: $\Sigma_{i=1}^{n} w_i b_i \leq k$ where $w_i, k \in \mathbb{Z}$)

- There are several CNF encoding techniques for these constraint types
  - Pairwise encoding (AMO), sequential counter, bit blasting, sorting networks, BDDs, watchdog encodings, ...

Cardinality Constraints

Presentation topic

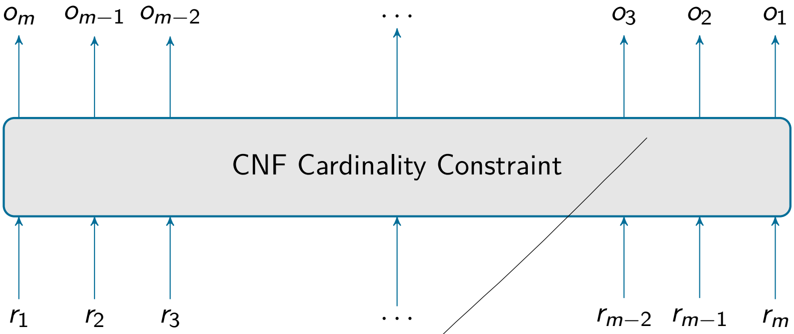
Is there a solution of the hard clauses that falsifies **at most** k soft clauses?

- Cardinality constraints must be encoded as CNF:

$$\Sigma_{i=1}^{n} b_i \bowtie k \text{ where } \bowtie \in \{\leq, =, \geq\}$$

  - At-Most-1 (AMO) constraints: $\Sigma_{i=1}^{n} b_i \leq 1$
  - General cardinality constraints: $\Sigma_{i=1}^{n} b_i \leq k$
  - (Pseudo-Boolean constraints: $\Sigma_{i=1}^{n} w_i b_i \leq k$ where $w_i, k \in \mathbb{Z}$)
- There are several CNF encoding techniques for these constraint types
  - Pairwise encoding (AMO), sequential counter, bit blasting, sorting networks, BDDs, watchdog encodings, ...
- Sequences of relaxations of such cardinality constraints suggest the need of incremental cardinality constraints

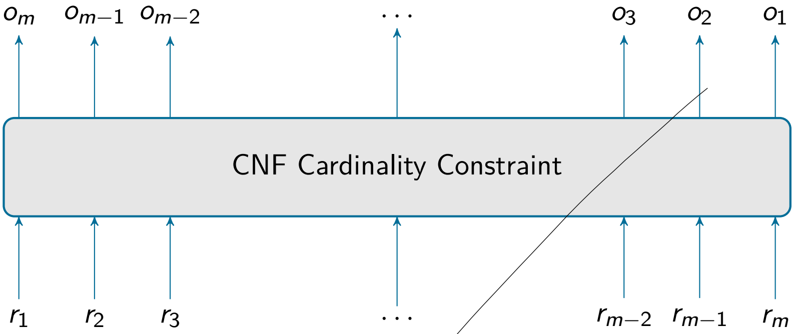
Cardinality Constraints – Totalizers



The clauses of the encoding ensure that

- $o_m o_{m-1} \dots o_2 o_1$: unary representation of the sum of the variables $r_1, r_2, \dots, r_m$

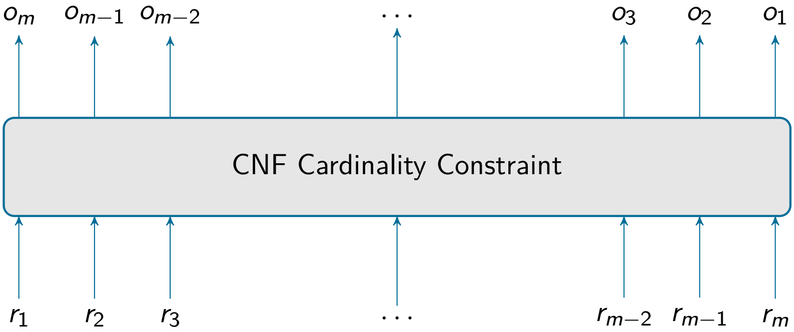
Cardinality Constraints – Totalizers



The clauses of the encoding ensure that

- $o_m o_{m-1} \dots o_2 o_1$: unary representation of the sum of the variables $r_1, r_2, \dots, r_m$
- $o_i \leftrightarrow r_1 + r_2 + \dots + r_k \geq i$
- $\neg o_i \leftrightarrow r_1 + r_2 + \dots + r_k < i$

Cardinality Constraints – Totalizers



The clauses of the encoding ensure that

- $o_m o_{m-1} \ldots o_2 o_1$: unary representation of the sum of the variables $r_1, r_2, \ldots, r_m$
- $o_i \leftrightarrow r_1 + r_2 + \ldots + r_k \geq i$
- $\neg o_i \leftrightarrow r_1 + r_2 + \ldots + r_k < i$

Most often used in MaxSAT: Incremental Totalizer Encoding: [MartinJML-CP'14]

Fu-Malik Algorithm (2006)

Algorithm 3 Fu-Malik Algorithm [FuMalik-SAT'06]

$F^b = \text{bv\_transform}(F)$; card_layer $= \{\}$; A $= \{(b_i) \mid (b_i) \in \mathit{soft}(F^b)\}$
$F' = \mathit{hard}(F^b)$; cost $= 0$; sat? $=$ *false*
while not sat? **do**
    (sat?,τ,κ) $\leftarrow$ $\texttt{SAT}(F' \cup$ card_layer, assume $= A)$
    if not sat? **then**
        for all $\neg b_i \in \kappa$ **do**
            $C_i \leftarrow C_i \vee r_i^{cost}$
        card_layer $\leftarrow$ card_layer $\cup$ $\text{CNF}(\Sigma_{\{i \mid \neg b_i \in \kappa\}} r_i^{cost} = 1)$
        cost $\leftarrow$ cost $+1$
    else
        return $(\tau \mid_{vars(F)}, \text{cost})$

Fu-Malik Algorithm Example

Project Idea

$$F = (x \vee \neg y) \wedge (y \vee z) \wedge (y \vee \neg z) \wedge (\neg x)_1 \wedge (\neg y)_1$$

$$F^b = (x \vee \neg y) \wedge (y \vee z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg b_1) \wedge (\neg y \vee \neg b_2) \wedge (b_1)_1 \wedge (b_2)_1$$

1. $\texttt{SAT}(F^b$,assume$=\{b_1, b_2\})$
   - UNSAT, $\kappa = (\neg b_1 \vee \neg b_2)$
   - Extend $(\neg x \vee \neg b_1)$ to $(\neg x \vee r_1^0 \vee \neg b_1)$
     and $(\neg y \vee \neg b_2)$ to $(\neg y \vee r_2^0 \vee \neg b_2)$
   - Add cardinality constraint CNF$(r_1^0 + r_2^0 = 1)$
2. $\texttt{SAT}(F^b \cup$CNF$(r_1^0 + r_2^0 = 1)$,assume$=\{b_1, b_2\})$
   - UNSAT, $\kappa = (\neg b_1 \vee \neg b_2)$
   - Extend $(\neg x \vee r_1^0 \vee \neg b_1)$ to $(\neg x \vee r_1^0 \vee r_1^1 \vee \neg b_1)$
     and $(\neg y \vee r_2^0 \vee \neg b_2)$ to $(\neg y \vee r_2^0 \vee r_2^1 \vee \neg b_2)$
   - Add cardinality constraint CNF$(r_1^1 + r_2^1 = 1)$
3. $\texttt{SAT}(F^b \cup$CNF$(r_1^0 + r_2^0 = 1) \cup$CNF$(r_1^1 + r_2^1 = 1)$,assume$=\{b_1, b_2\})$
   - SAT, cost$(\tau) = 2$

Fu-Malik Algorithm Remarks

- – Only works on unweighted MaxSAT problems (soft clause cloning is used to extend it to unweighted, see WPM1 [AnsóteguiBonetLevy-SAT'09] and WMSU1 [ManquinhoMarques-SilvaPlanes-SAT'09] algorithms)
- – Changes F^b by extending clauses with new variables $\rightarrow$ not really incremental SAT friendly
- – Multiple relaxation variables in same clauses can lead to symmetries $\rightarrow$ can slow down SAT solving
- + Size of cardinality constraints is smaller (depends only on size of found cores, not on number of soft clauses)
- + Relaxing clauses only on demand

MSU3 Algorithm

Project idea

Algorithm 4 MSU3 Algorithm for Unweighted MaxSAT [Marques-SPlanes-CoRR'07]

$F^b = \text{bv\_transform}(F)$; card_layer $= \{\}$; A $= \{(b_i) \mid (b_i) \in \mathit{soft}(F^b)\}$
inCard $= \{\}$
$F' = \mathit{hard}(F^b)$; cost $= 0$; sat? $=$ *false*
while not sat? **do**
  (sat?,τ,κ) $\leftarrow$ $\mathtt{SAT}(F' \cup$ card_layer, assume $= A)$
  if not sat? **then**
    inCard $\leftarrow$ inCard $\cup\ \kappa$
    $A \leftarrow A \setminus \{b_i \mid \neg b_i \in \kappa\}$
    cost $\leftarrow$ cost $+1$
    card_layer $\leftarrow \text{CNF}(\Sigma_{\neg b_i \in inCard} \neg b_i \leq \mathit{cost})$
  else
    return $(\tau\ |_{vars(F)}, \text{cost})$

1st constraint

$r_1 + r_2 + r_3 + r_4 \leq 2$

2nd constraint

MSU3 Algorithm Example

$$F = (x \vee \neg y) \wedge (y \vee z) \wedge (y \vee \neg z) \wedge (\neg x)_1 \wedge (\neg y)_1$$

$$F^b = (x \vee \neg y) \wedge (y \vee z) \wedge (y \vee \neg z) \wedge (\neg x \vee \neg b_1) \wedge (\neg y \vee \neg b_2) \wedge (b_1)_1 \wedge (b_2)_1$$

1. $\texttt{SAT}(F^b$,assume$=\{b_1, b_2\})$
   - UNSAT, $\kappa = (\neg b_1 \vee \neg b_2)$
   - Add $\neg b_1$ and $\neg b_2$ to *inCard*
   - Remove b_1 and b_2 from A
   - Add cardinality constraint CNF$(\neg b_1 + \neg b_2 \leq 1)$ to card_layer
2. $\texttt{SAT}(F^b \cup$CNF$(\neg b_1 + \neg b_2 \leq 1)$,assume$=\{\})$
   - UNSAT, $\kappa = \{\}$
   - inCard and A are unchanged
   - card_layer is updated to be CNF$(\neg b_1 + \neg b_2 \leq 2)$
3. $\texttt{SAT}(F^b \cup$CNF$(\neg b_1 + \neg b_2 \leq 2)$,assume$=\{\})$
   - SAT, cost $= 2$

MSU3 Algorithm Remarks

+ Cardinality constraint changes in a systematic way: LHS gets new variables (from a predefined set of variables) and RHS is always incremented
  - → fully incremental SAT queries through totalizer encoding [MartinsJoshiManquinhoLynce-CP'14]
- One cardinality constraint over union of ALL cores
  - → can be decomposed into smaller constraints over disjoint cores (see PM2 algorithm [AnsóteguiBonetLevy-AI'13])

OLL Algorithm [MorgadoDodaroMarques'14]

- Use "soft cardinality constraints"
- Further exploits incremental totalizer encoding (output variables of cardinality constraints are used as new soft clauses)
  $\rightarrow$ SAT solver can find cores over cardinality constraints
- One of the state-of-the-art approaches currently
- Efficiently implemented by RC2 [IgnatievMorgadoMarques-JSAT'18]

The Implicit Hitting Set Approach

Hitting Sets

Definition (Hitting Set)

Let K be a set of cores, i.e., a set of sets of soft clauses. A hitting set η of K is a set of soft clauses that has a non-empty intersection with every set in K:

$$\forall \kappa \in K : \eta \cap \kappa \neq \emptyset$$

Hitting Sets

Definition (Hitting Set)

Let K be a set of cores, i.e., a set of sets of soft clauses. A hitting set η of K is a set of soft clauses that has a non-empty intersection with every set in K:

$$\forall \kappa \in K : \eta \cap \kappa \neq \emptyset$$

$$\mathcal{F} = (\neg x_1 \vee \neg x_2)_1 \wedge (\neg x_2 \vee x_3)_1 \wedge (\neg x_3 \vee \neg x_4)_1 \wedge (x_1)_1 \wedge (x_2)_1 \wedge (x_4)_1$$

$$\kappa_0 = \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_1)_1, (x_2)_1, (x_4)_1\}$$

$$\kappa_1 = \{(\neg x_1 \vee \neg x_2)_1, (x_1)_1, (x_2)_1\}$$

$$\kappa_2 = \{(\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_2)_1, (x_4)_1\}$$

Hitting Sets

Definition (Hitting Set)

Let K be a set of cores, i.e., a set of sets of soft clauses. A hitting set η of K is a set of soft clauses that has a non-empty intersection with every set in K:

$$\forall \kappa \in K : \eta \cap \kappa \neq \emptyset$$

$$\mathcal{F} = (\neg x_1 \vee \neg x_2)_1 \wedge (\neg x_2 \vee x_3)_1 \wedge (\neg x_3 \vee \neg x_4)_1 \wedge (x_1)_1 \wedge (x_2)_1 \wedge (x_4)_1$$

$$\kappa_0 = \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_1)_1, (x_2)_1, (x_4)_1\}$$

$$\kappa_1 = \{(\neg x_1 \vee \neg x_2)_1, (x_1)_1, (x_2)_1\}$$

$$\kappa_2 = \{(\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_2)_1, (x_4)_1\}$$

$$\mathrm{HS}(\kappa_0, \kappa_1, \kappa_2) : \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1\} \quad \mathrm{cost} \sum : 2$$

Hitting Sets

Definition (Hitting Set)

Let K be a set of cores, i.e., a set of sets of soft clauses. A hitting set η of K is a set of soft clauses that has a non-empty intersection with every set in K:

$$\forall \kappa \in K : \eta \cap \kappa \neq \emptyset$$

$$\mathcal{F} = (\neg x_1 \vee \neg x_2)_1 \wedge (\neg x_2 \vee x_3)_1 \wedge (\neg x_3 \vee \neg x_4)_1 \wedge (x_1)_1 \wedge (x_2)_1 \wedge (x_4)_1$$

$$\kappa_0 = \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_1)_1, (x_2)_1, (x_4)_1\}$$
$$\kappa_1 = \{(\neg x_1 \vee \neg x_2)_1, (x_1)_1, (x_2)_1\}$$
$$\kappa_2 = \{(\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_2)_1, (x_4)_1\}$$

$$\mathrm{HS}(\kappa_0, \kappa_1, \kappa_2) : \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1\} \quad \mathrm{cost} \sum : 2$$

- Minimum Cost Hitting Set: Hitting set with cost less than or equal to the cost of any other hitting set

Hitting Sets

Definition (Hitting Set)

Let K be a set of cores, i.e., a set of sets of soft clauses. A hitting set η of K is a set of soft clauses that has a non-empty intersection with every set in K:

$$\forall \kappa \in K : \eta \cap \kappa \neq \emptyset$$

$$\mathcal{F} = (\neg x_1 \vee \neg x_2)_1 \wedge (\neg x_2 \vee x_3)_1 \wedge (\neg x_3 \vee \neg x_4)_1 \wedge (x_1)_1 \wedge (x_2)_1 \wedge (x_4)_1$$

$$\kappa_0 = \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_1)_1, (x_2)_1, (x_4)_1\}$$
$$\kappa_1 = \{(\neg x_1 \vee \neg x_2)_1, (x_1)_1, (x_2)_1\}$$
$$\kappa_2 = \{(\neg x_2 \vee x_3)_1, (\neg x_3 \vee \neg x_4)_1, (x_2)_1, (x_4)_1\}$$

$$\mathrm{HS}(\kappa_0, \kappa_1, \kappa_2) : \{(\neg x_1 \vee \neg x_2)_1, (\neg x_2 \vee x_3)_1\} \quad \mathrm{cost} \sum : 2$$

- Minimum Cost Hitting Set: Hitting set with cost less than or equal to the cost of any other hitting set

$$\mathrm{MinHS}(\kappa_0, \kappa_1, \kappa_2) : \{(x_2)_1\} \quad \mathrm{cost} \sum : 1$$

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS$(U) \leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS$(U) \leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS$(U) \leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.
Approach: Calculate a hitting set η for the already known unsatisfiable cores (K).

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS$(U) \leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.
Approach: Calculate a hitting set η for the already known unsatisfiable cores (K).

- If $\{F - \eta\}$ is UNSAT: new core κ can be added to K.

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS(U) $\leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.
Approach: Calculate a hitting set η for the already known unsatisfiable cores (K).

- If $\{F - \eta\}$ is UNSAT: new core κ can be added to K.
- If $\{F - \eta\}$ is SAT:

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS(U) $\leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.
Approach: Calculate a hitting set η for the already known unsatisfiable cores (K).

- If $\{F - \eta\}$ is UNSAT: new core κ can be added to K.
- If $\{F - \eta\}$ is SAT:
  - If η is MinHS: Any satisfying assignment is guaranteed to be optimal.

Hitting Sets & MaxSAT [DaviesBacchus-CP'11]

To solve a MaxSAT problem F it is enough to find a minimum cost hitting set η of **all** the unsatisfiable cores (U) of F.

- $\forall \kappa \in U : \eta \cap \kappa \neq \emptyset \leftrightarrow \{F - \eta\}$ is SAT
- η is MinHS$(U) \leftrightarrow$ all satisfying assignments of $\{F - \eta\}$ is optimal

Issue: We do not know all the unsatisfiable cores in advance.
Approach: Calculate a hitting set η for the already known unsatisfiable cores (K).

- If $\{F - \eta\}$ is UNSAT: new core κ can be added to K.
- If $\{F - \eta\}$ is SAT:
  - If η is MinHS: Any satisfying assignment is guaranteed to be optimal.
  - If η is arbitrary HS: No guarantee of optimality.

IHS Algorithms for MaxSAT

$$\mathcal{F}$$

$$\left\{ \begin{array}{ccc} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & x_1 \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

IHS Algorithms for MaxSAT

$$\mathcal{F}$$

$$\left\{ \begin{array}{ccc} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & x_1 \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

IHS Algorithms for MaxSAT

$\mathcal{F}$

$$\left\{ \begin{array}{ccc} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & x_1 \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

$\mathrm{HS}(K)$

$$\left\{ \quad \right\}$$

cost over *soft*($\mathcal{F}$) $\xrightarrow{K = \emptyset}$ OPT

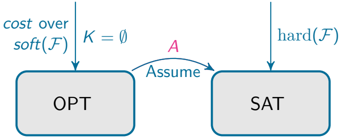
SAT

IHS Algorithms for MaxSAT

$$\mathcal{F} \qquad\qquad\qquad\qquad\qquad\qquad \mathrm{HS}(K)$$

$$\left\{ \begin{array}{ccc} (\neg x_1 \lor \neg x_2 \lor \neg b_1) & b_1 & x_1 \\ (\neg x_2 \lor x_3 \lor \neg b_2) & b_2 & x_2 \\ (\neg x_3 \lor \neg x_4 \lor \neg b_3) & b_3 & x_4 \end{array} \right\} \quad \left\{ \quad \right\}$$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \text{assume} = \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )?$

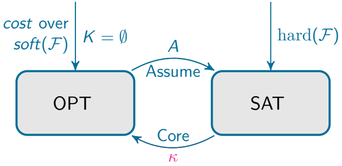


IHS Algorithms for MaxSAT

$$\mathcal{F} \qquad\qquad\qquad\qquad \mathrm{HS}(K)$$

$$\left\{ \begin{array}{lll} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & x_1 \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\} \qquad \left\{ \quad \right\}$$

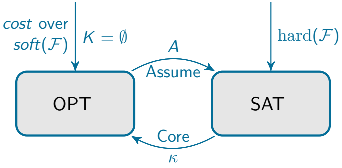
$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F})$, assume $= \{\, b_1, b_2, b_3, x_1, x_2, x_4 \,\}\ )$?



IHS Algorithms for MaxSAT

$$\mathcal{F} = \left\{ \begin{array}{ccc} (\neg x_1 \lor \neg x_2 \lor \neg b_1) & b_1 & \\ (\neg x_2 \lor x_3 \lor \neg b_2) & b_2 & x_2 \\ (\neg x_3 \lor \neg x_4 \lor \neg b_3) & b_3 & x_4 \end{array} \right\} \qquad \mathrm{HS}(K) = \left\{ x_1 \right\}$$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F})$, assume $= \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )$?

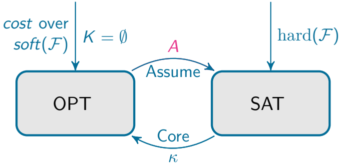


IHS Algorithms for MaxSAT

$$\mathcal{F} = \left\{ \begin{array}{lll} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\} \qquad \mathrm{HS}(K) = \left\{ \begin{array}{c} x_1 \\ \\ \\ \end{array} \right\}$$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \text{assume} = \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )?$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \text{assume} = \{\ b_1, b_2, b_3, \quad x_2, x_4\ \}\ )?$



IHS Algorithms for MaxSAT

$\mathcal{F}$

$$\left\{ \begin{array}{lll} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

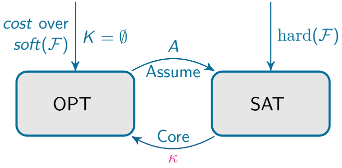
HS(K)

$$\left\{ \; x_1 \; \right\}$$

(2nd unsat core

$\texttt{SAT}($ hard$(\mathcal{F})$, assume $= \{ b_1, b_2, b_3, x_1, x_2, x_4 \}$ $)$?

$\texttt{SAT}($ hard$(\mathcal{F})$, assume $= \{ b_1, b_2, b_3, x_2, x_4 \}$ $)$?



IHS Algorithms for MaxSAT

$\mathcal{F}$

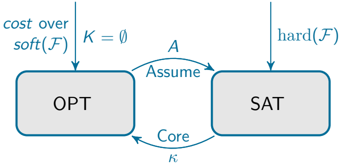
$$\left\{ \begin{array}{lll} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & \\ (\neg x_2 \vee x_3 \vee \neg b_2) & & x_2 \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

$\mathrm{HS}(K)$

$$\left\{ \begin{array}{c} x_1 \\ b_2 \\ \end{array} \right\}$$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )?$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, \quad x_2, x_4\ \}\ )?$



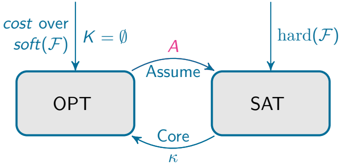
IHS Algorithms for MaxSAT

$$\mathcal{F} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_2 \vee \neg b_1) \\ (\neg x_2 \vee x_3 \vee \neg b_2) \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) \end{array} \right\} \qquad \left\{ \begin{array}{ll} b_1 & \\ & x_2 \\ b_3 & x_4 \end{array} \right\} \qquad \mathrm{HS}(K) = \left\{ \begin{array}{l} x_1 \\ b_2 \\ \\ \end{array} \right\}$$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )?$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, x_2, x_4\ \}\ )?$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_3, x_2, x_4\ \}\ )?$



IHS Algorithms for MaxSAT

$$\mathcal{F} \quad \left\{ \begin{array}{ll} (\neg x_1 \lor \neg x_2 \lor \neg b_1) & b_1 \\ (\neg x_2 \lor x_3 \lor \neg b_2) & \quad\quad x_2 \\ (\neg x_3 \lor \neg x_4 \lor \neg b_3) & b_3 \quad\quad x_4 \end{array} \right\} \qquad \mathrm{HS}(K) \quad \left\{ \begin{array}{c} x_1 \\ b_2 \\ \\ \end{array} \right\}$$

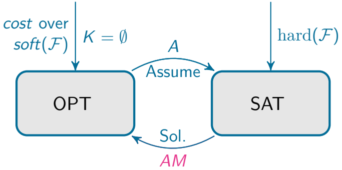
$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{ b_1, b_2, b_3, x_1, x_2, x_4 \}\ )$?

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{ b_1, b_2, b_3, x_2, x_4 \}\ )$?

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{ b_1, b_3, x_2, x_4 \}\ )$?

feasible solution since it was an arbitrary hitting set

→ otherwise optimal



IHS Algorithms for MaxSAT

$\mathcal{F}$

$$\left\{ \begin{array}{lll} (\neg x_1 \vee \neg x_2 \vee \neg b_1) & b_1 & x_1 \\ (\neg x_2 \vee x_3 \vee \neg b_2) & b_2 & \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) & b_3 & x_4 \end{array} \right\}$$

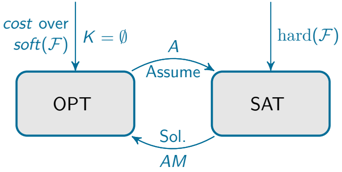
MinHS(K)

$$\left\{ \; x_2 \; \right\}$$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, x_1, x_2, x_4\ \}\ )?$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, b_2, b_3, \quad x_2, x_4\ \}\ )?$

$\texttt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\ b_1, \quad b_3, \quad x_2, x_4\ \}\ )?$

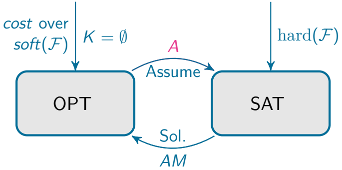


IHS Algorithms for MaxSAT

$$\mathcal{F} \qquad\qquad\qquad\qquad \mathrm{MinHS}(K)$$

$$\left\{ \begin{array}{c} (\neg x_1 \vee \neg x_2 \vee \neg b_1) \\ (\neg x_2 \vee x_3 \vee \neg b_2) \\ (\neg x_3 \vee \neg x_4 \vee \neg b_3) \end{array} \quad \begin{array}{cc} b_1 & x_1 \\ b_2 & \\ b_3 & x_4 \end{array} \right\} \qquad \left\{ \begin{array}{c} \\ x_2 \\ \\ \end{array} \right\}$$

$$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \textsf{assume} = \{\, b_1, b_2, b_3, x_1, x_2, x_4 \,\}\ )?$$
$$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \textsf{assume} = \{\, b_1, b_2, b_3, x_2, x_4 \,\}\ )?$$
$$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \textsf{assume} = \{\, b_1, b_3, x_2, x_4 \,\}\ )?$$
$$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \textsf{assume} = \{\, b_1, b_2, b_3, x_1, x_4 \,\}\ )?$$



IHS Algorithms for MaxSAT

hard const. soft in blue red ⟺ core

$$\mathcal{F}: \left\{ \begin{array}{lll} (\neg x_1 \lor \neg x_2 \lor \neg b_1) & b_1 & x_1 \\ (\neg x_2 \lor x_3 \lor \neg b_2) & b_2 & \\ (\neg x_3 \lor \neg x_4 \lor \neg b_3) & b_3 & x_4 \end{array} \right\} \qquad \mathrm{MinHS}(K): \left\{ \; x_2 \; \right\}$$

1st query

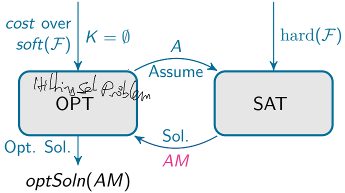
1.) $\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\, b_1, b_2, b_3, x_1, x_2, x_4 \,\}\ )?$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\, b_1, b_2, b_3, x_2, x_4 \,\}\ )?$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\, b_1, b_3, x_2, x_4 \,\}\ )?$

$\mathtt{SAT}(\ \mathrm{hard}(\mathcal{F}),\ \mathsf{assume} = \{\, b_1, b_2, b_3, x_1, x_4 \,\}\ )?$

↳ the [illegible] optimal solution



Appendix

OLL Algorithm for MaxSAT

Algorithm 5: OLL algorithm for MaxSAT [MDM14]

OLL(F)
  F^b = bv_transform(F); card_layer = {}; assumptions = $\{b_i \mid (b_i) \in soft(F^b)\}$
  $F' = hard(F^b)$; cost = 0; iter = 0; sat? = *false*
  while *not sat?* **do**
    /* *SAT solver interface is the same as in Algorithm 2* */
    (sat?, π, κ) = **SATSolve** ($F' \cup$ card_layer, assumptions)
    if *not sat?* **then**
      $wt_{\min} = \min_{\{\ell \mid \ell \in \kappa\}} wt((\neg \ell))$
      cost = cost + $wt_{\min}$
      for $\ell \in \kappa$ **do** /* $(\neg \ell)$ *is a soft clause* */
        $wt((\neg \ell)) = wt((\neg \ell)) - wt_{\min}$
        if $wt((\neg \ell)) == 0$ **then**
          assumptions = assumptions $\setminus \{\neg \ell\}$
      for $o^i_j \in \kappa$ **do** /* *Summation output variables* */
        if $wt((\neg o^i_j)) == 0 \wedge o^i_{j+1}$ *exists* **then**
          /* *i'th summation has another output variable* */
          assumptions = assumptions $\cup \{\neg o^i_{j+1}\}$
      /* *Encode clauses that make output variables equal to value of sum* */
      card_layer = card_layer $\cup$ CNF$(\sum_{\ell \in \kappa} \ell \geq j \equiv o^{\text{iter}}_j$ for $o^{\text{iter}}_2, \ldots, o^{\text{iter}}_{|\kappa|})$
      for $j = 2, \ldots, |\kappa|$ **do**
        $wt((\neg o^{\text{iter}}_j)) = wt_{\min}$
      assumptions = assumptions $\cup \{\neg o^{\text{iter}}_2\}$
      iter = iter + 1
    else
      return $(\pi|_{vars(F)}, \text{cost})$

DIMACS for Weighted CNF

Example (QDIMACS Format)

- Extension of DIMACS format used in SAT solving.
- Literals of variables encoded as signed integers.
- One clause per line, terminated by zero.
- header: line "p wcnf nbvar nbclauses top" where
  nbvar: number of variables
  nbclauses: number of clauses
  top: weight of hard clauses (optional)

```
p wcnf 4 5 15
15 1 -2 4 0
15 -1 -2 3 0
2 -2 -4 0
5 -3 2 0
3 1 3 0
```

WCNF DIMACS format: `http://www.maxhs.org/docs/wdimacs.html`