Incremental Maximum Satisfiability

Reporter: Chi Zhiming

December 14, 2022

目录

目录

Contribution

1. Detail **various forms of incrementality in MaxSAT**
    - adding hard clauses, soft literals, assumptions
2. Propose **IPAMIR: incremental API for MaxSAT**
    - generic interface for developing incremental MaxSAT solvers and applications making use of incremental MaxSAT
    - MaxSAT Evaluation 2022: incremental track
3. Develop a fully-fledged **incremental MaxSAT solver**
    - support for all functionality specified in IPAMIR
    - extends MaxHS: the state-of-the-art implicit hitting set based solver
4. Provide **empirical evidence on benefits of incrementality**
    - solving under different sets of assumptions

Contribution

1. Detail **various forms of incrementality in MaxSAT**
   - adding hard clauses, soft literals, assumptions
2. Propose **IPAMIR: incremental API for MaxSAT**
   - generic interface for developing incremental MaxSAT solvers and applications making use of incremental MaxSAT
   - MaxSAT Evaluation 2022: incremental track
3. Develop a fully-fledged **incremental MaxSAT solver**
   - support for all functionality specified in IPAMIR
   - extends MaxHS: the state-of-the-art implicit hitting set based solver
4. Provide **empirical evidence on benefits of incrementality**
   - solving under different sets of assumptions

Contribution

1. Detail **various forms of incrementality in MaxSAT**
    - adding hard clauses, soft literals, assumptions
2. Propose **IPAMIR: incremental API for MaxSAT**
    - generic interface for developing incremental MaxSAT solvers and applications making use of incremental MaxSAT
    - MaxSAT Evaluation 2022: incremental track
3. Develop a fully-fledged **incremental MaxSAT solver**
    - support for all functionality specified in IPAMIR
    - extends MaxHS: the state-of-the-art implicit hitting set based solver
4. Provide **empirical evidence on benefits of incrementality**
    - solving under different sets of assumptions

Contribution

1. Detail **various forms of incrementality in MaxSAT**
   - adding hard clauses, soft literals, assumptions
2. Propose **IPAMIR: incremental API for MaxSAT**
   - generic interface for developing incremental MaxSAT solvers and applications making use of incremental MaxSAT
   - MaxSAT Evaluation 2022: incremental track
3. Develop a fully-fledged **incremental MaxSAT solver**
   - support for all functionality specified in IPAMIR
   - extends MaxHS: the state-of-the-art implicit hitting set based solver
4. Provide **empirical evidence on benefits of incrementality**
   - solving under different sets of assumptions

Incremental Optimization

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - adding or removing constraints
  - modifying objective function
- Solving each instance from scratch often too costly: **reuse information obtained during previous calls**
- Incremental SAT solving well-established
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental Optimization

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - adding or removing constraints
  - modifying objective function
- Solving each instance from scratch often too costly: **reuse information obtained during previous calls**
- Incremental SAT solving well-established
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental Optimization

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - adding or removing constraints
  - modifying objective function
- Solving each instance from scratch often too costly: **reuse information obtained during previous calls**
- Incremental SAT solving well-established
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental Optimization

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - adding or removing constraints
  - modifying objective function
- Solving each instance from scratch often too costly: **reuse information obtained during previous calls**
- Incremental SAT solving well-established
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality

MaxSAT

- Optimization extension of SAT
- An instance consists of
  - a set of hard clauses $\mathcal{F}_H$,
  - a set of soft literals S,
  - a weight function w over soft literals S.
- Find τ that satisfies all hard clauses and minimizes $\sum_{b \in S} w(b) \cdot b$.
- Definition equivalent to weighted soft clauses $\mathcal{F}_L$:
  - relax each soft clause $C \in \mathcal{F}_L$ to $C \vee b_C$.
  - add $C \vee b_C$ to $\mathcal{F}_H$, and $\neg b_C$ to S with weight $w(b_C) = w(C)$.
- Assumptions A: A set of literals
- MaxSAT instance $\mathcal{F}$ under a set of assumptions:
  - $\mathcal{F} \wedge A = (\mathcal{F}_H \wedge \bigwedge_{l \in A}(l), \mathcal{F}_L, w)$

MaxSAT

- Optimization extension of SAT
- An instance consists of
  - a set of hard clauses $\mathcal{F}_H$,
  - a set of soft literals S,
  - a weight function w over soft literals S.
- Find τ that satisfies all hard clauses and minimizes $\sum_{b \in S} w(b) \cdot b$.
- Definition equivalent to weighted soft clauses $\mathcal{F}_L$:
  - relax each soft clause $C \in \mathcal{F}_L$ to $C \vee b_C$.
  - add $C \vee b_C$ to $\mathcal{F}_H$, and $\neg b_C$ to S with weight $w(b_C) = w(C)$.
- Assumptions A: A set of literals
- MaxSAT instance $\mathcal{F}$ under a set of assumptions:
  - $\mathcal{F} \wedge A = \left(\mathcal{F}_H \wedge \bigwedge_{l \in A}(l), \mathcal{F}_L, w\right)$

Example 1

- **Example 1.** Consider the MaxSAT instance $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_L, w)$ with $\mathcal{F}_H = \{(b_1 \vee x), (\neg x \vee b_2), (\neg z), (z \vee y \vee b_3 \vee b_4), (\neg y \vee b_3 \vee b_4)\}$, $\mathcal{F}_L = \{b_1, b_2, b_3, b_4\}$ and $w(b_1) = w(b_3) = w(b_4) = 1$ and $w(b_2) = 2$. The solution $\tau = \{b_1, \neg b_2, b_3, \neg b_4, \neg x, y, \neg z\}$ is an optimal solution of $\mathcal{F}$, with $\textsc{cost}(\mathcal{F}, \tau) = \textsc{cost}(\mathcal{F}) = 2$.

Incremental changes in MaxSAT

- Aim for solving a sequence of related MaxSAT instances efficiently, avoiding computation from scratch
- Different forms of incremental changes:

- *Adding hard clauses*. Given a clause C, add it to the hard clauses of $\mathcal{F}$: $\textsc{addHard}(\mathcal{F}, C) = (\mathcal{F}_H \cup \{C\}, \mathcal{F}_L, w)$.
- *Adding soft literals*. Given a variable $b \notin \mathcal{F}_L$ with weight w_b, add it to the set of soft literals of $\mathcal{F}$: $\textsc{addSoft}(\mathcal{F}, b, w_b) = (\mathcal{F}_H, \mathcal{F}_L \cup \{b\}, w \cup \{b \mapsto w_b\})$.
- *Changing the weight of a soft literal.* Given a literal $b \in \mathcal{F}_L$ with weight w_b, change its weight in $\mathcal{F}$ to w_b: $\textsc{changeWeight}(\mathcal{F}, b, w_b) = (\mathcal{F}_H, \mathcal{F}_L, (w \setminus \{b \mapsto w(b)\}) \cup \{b \mapsto w_b\})$.

Incremental changes in MaxSAT

- Different scenarios call for different forms of incremental changes
  - adding hard clauses: MaxSAT-based CEGAR
    Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]
  - changing weights of soft literals: AdaBoost
    Hu, Siala, Hebrard, and Huguet [2020]
  - solving under assumptions: timetabling with disruptions
    Lemos, Monteiro, and Lynce [2020]
- Assumptions can be used to simulate the removal of clauses and hardening soft clauses.

Example 2

- **Example 2.** Consider the MaxSAT instance $\mathcal{F}$ from Example 1. Suppose we solve it under the assumptions $A = \{x\}$, that is, enforcing that $\tau(x) = 1$ must hold for any solution of $\mathcal{F}$. Now $\tau = \{\neg b_1, b_2, b_3, \neg b_4, x, y, \neg z\}$ is an optimal solution of $\mathcal{F}$ under the assumptions A, with $\textsc{cost}(\mathcal{F} \wedge A, \tau) = 3$. Note that if we view the hard clause $b_1 \vee x$ as a normalized soft clause x with weight $w(b_1)$, by assuming $\{\neg b_1\}$ we effectively *harden* the soft clause x, which in this case achieves the same result as assuming $\{x\}$. Now suppose we instead consider the instance $\mathcal{F}' = \textsc{changeWeight}(\mathcal{F}, b_1, 0)$ which sets the weight of the first soft literal to zero. An optimal solution of $\mathcal{F}'$ is $\tau' = \{b_1, \neg b_2, b_3, \neg b_4, \neg x, y, \neg z\}$ with cost $\textsc{cost}(\mathcal{F}', \tau') = 1$. Note how assigning its weight to 0 effectively removes b_1 as a soft literal.

目录

IPAMIR: Interface for Incremental MaxSAT Solving

- **Generic interface** for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on **IPASIR**: standard interface for incremental SAT
- Specifies **incremental changes** to a MaxSAT instance
  - adding hard clauses
  - adding soft literals or changing their weights
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Interface for Incremental MaxSAT Solving

- **Generic interface** for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on **IPASIR**: standard interface for incremental SAT
- Specifies **incremental changes** to a MaxSAT instance
  - adding hard clauses
  - adding soft literals or changing their weights
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Interface for Incremental MaxSAT Solving

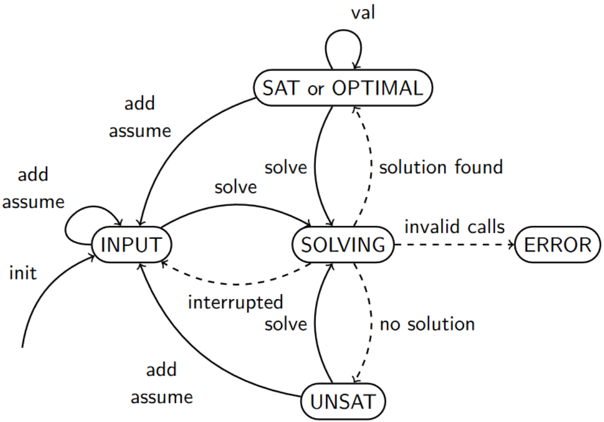
- **Generic interface** for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on **IPASIR**: standard interface for incremental SAT
- Specifies **incremental changes** to a MaxSAT instance
  - adding hard clauses
  - adding soft literals or changing their weights
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Interface for Incremental MaxSAT Solving

- **Generic interface** for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on **IPASIR**: standard interface for incremental SAT
- Specifies **incremental changes** to a MaxSAT instance
  - adding hard clauses
  - adding soft literals or changing their weights
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

```
// Construct a MaxSAT solver and return a pointer to it.
void * ipamir_init ();
// Deallocate all resources of the MaxSAT solver.
void ipamir_release (void * solver);
// Add a literal to a hard clause or finalize the clause with zero.
void ipamir_add_hard (void * solver, int32_t lit_or_zero);
// Add a weighted soft literal.
void ipamir_add_soft_lit (void * solver, int32_t lit, uint64_t weight);
// Assume a literal for the next solver call.
void ipamir_assume (void * solver, int32_t lit);
// Solve the MaxSAT instance under the current assumptions.
int ipamir_solve (void * solver);
// Compute the cost of the solution.
uint64_t ipamir_val_obj (void * solver);
// Extract the truth value of a literal in the solution.
int32_t ipamir_val_lit (void * solver, int32_t lit);
// Set a callback function for terminating the solving procedure.
void ipamir_set_terminate (void * solver, void * state,
                           int (*terminate)(void * state));
```

IPAMIR: Interface for Incremental MaxSAT Solving



目录

Implicit Hitting Set (IHS) based MaxSAT solving

- An iterative approach: identify sources of inconsistency and repair the inconsistencies in a minimal way.
  - A (unsat)core is a clause over soft literals entailed by the hard clauses.
    - SAT solver as core extractor
  - hs is a hitting set over a set of cores $\mathcal{C}$ if hs intersects each $\kappa \in \mathcal{C}$
    - cost of a hitting set determined by weights of soft literals
    - IP solver for computing minimum-cost hitting sets
- Reasoning and optimization effectively decoupled:
  - upper bounds from assignments given by the SAT solver
  - lower bounds from costs of optimal hitting sets

Implicit Hitting Set (IHS) based MaxSAT solving

- An iterative approach: identify sources of inconsistency and repair the inconsistencies in a minimal way.
  - A (unsat)core is a clause over soft literals entailed by the hard clauses.
    - SAT solver as core extractor
  - hs is a hitting set over a set of cores $\mathcal{C}$ if hs intersects each $\kappa \in \mathcal{C}$
    - cost of a hitting set determined by weights of soft literals
    - IP solver for computing minimum-cost hitting sets
- Reasoning and optimization effectively decoupled:
  - upper bounds from assignments given by the SAT solver
  - lower bounds from costs of optimal hitting sets

Implicit Hitting Set (IHS) based MaxSAT solving

- An iterative approach: identify sources of inconsistency and repair the inconsistencies in a minimal way.
    - A (unsat)core is a clause over soft literals entailed by the hard clauses.
        - SAT solver as core extractor
    - hs is a hitting set over a set of cores $\mathcal{C}$ if hs intersects each $\kappa \in \mathcal{C}$
        - cost of a hitting set determined by weights of soft literals
        - IP solver for computing minimum-cost hitting sets
- Reasoning and optimization effectively decoupled:
    - upper bounds from assignments given by the SAT solver
    - lower bounds from costs of optimal hitting sets

Implicit Hitting Set (IHS) based MaxSAT solving

Algorithm 1 MaxSAT solving via implicit hitting sets.

IHS($\mathcal{F}$)
- **Input:** An instance $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_L, w)$
- **Output:** An optimal solution τ
- $lb \leftarrow 0$; $ub \leftarrow \infty$;
- $\tau_{best} \leftarrow \varnothing$; $\mathcal{C} \leftarrow \emptyset$;
- **while** $(\textsc{true})$ **do**
  - $hs \leftarrow$ Min-Hs$(\mathcal{F}_L, \mathcal{C})$;
  - $lb = \textsc{cost}(\mathcal{F}, hs)$;
  - **if** $(lb = ub)$ **then break**;
  - $(K, \tau) \leftarrow$ Extract-Cores$(\mathcal{F}_H, \mathcal{F}_L, hs)$;
  - **if** $(\textsc{cost}(\mathcal{F}, \tau) < ub)$ **then**
    - $\tau_{best} \leftarrow \tau$; $ub \leftarrow \textsc{cost}(\mathcal{F}, \tau)$;
  - **if** $(lb = ub)$ **then return** τ_{best};
  - $\mathcal{C} \leftarrow \mathcal{C} \cup K$;

$$
\begin{array}{ll}
\textbf{minimize} & \displaystyle\sum_{b \in \mathcal{F}_L} w(b) \cdot b \\
\textbf{subject to} & \\
\sum_{b \in \kappa} b \geq 1 & \forall \kappa \in \mathcal{C} \\
b \in \{0, 1\} & \forall b \in \mathcal{F}_L
\end{array}
$$

Figure 3 An integer program for computing a hitting set over a set $\mathcal{C}$ of cores of an instance $\mathcal{F}$.

Example 1

- **Example 1.** Consider the MaxSAT instance $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_L, w)$ with $\mathcal{F}_H = \{(b_1 \vee x), (\neg x \vee b_2), (\neg z), (z \vee y \vee b_3 \vee b_4), (\neg y \vee b_3 \vee b_4)\}$, $\mathcal{F}_L = \{b_1, b_2, b_3, b_4\}$ and $w(b_1) = w(b_3) = w(b_4) = 1$ and $w(b_2) = 2$. The solution $\tau = \{b_1, \neg b_2, b_3, \neg b_4, \neg x, y, \neg z\}$ is an optimal solution of $\mathcal{F}$, with $\text{COST}(\mathcal{F}, \tau) = \text{COST}(\mathcal{F}) = 2$.

Example 3

- **Example 3.** Consider an invocation of `IHS` on the MaxSAT instance $\mathcal{F}$ from Example 1. Initially $\mathcal{C} = \emptyset$ so the first call to `Min-Hs` returns $hs = \emptyset$ which updates $lb = \text{COST}(\mathcal{F}, hs) = 0$. As $ub = \infty \neq 0 = lb$, the algorithm continues to the core extraction step. Assume the `Extract-Cores` subroutine returns $K = \{\{b_1, b_2\}, \{b_3, b_4\}\}$ and the solution $\tau = \{b_1, b_2, b_3, b_4, x, \neg y, \neg z\}$. The algorithm then updates $ub = \text{COST}(\mathcal{F}, \tau) = 5$. As $lb = 0 < 5 = ub$ the set K is added to $\mathcal{C}$ and the algorithm reiterates. In the next iteration $\mathcal{C} = \{\{b_1, b_2\}, \{b_3, b_4\}\}$ so `Min-Hs` computes (for example) the hitting set $hs = \{b_1, b_3\}$. The lower bound lb is then updated to $\text{COST}(\mathcal{F}, hs) = 2 < 5 = ub$ before invoking the next core extraction step. This time around, the first SAT solver call in `Extract-Cores` is done with the (solver) assumptions $\{\neg b \mid b \in \mathcal{F}_L \setminus hs\} = \{\neg b_2, \neg b_4\}$. The result is SAT, the solver returns the solution $\tau = \{b_1, b_3, \neg b_2, \neg b_4, \neg x, \neg z, y\}$. The procedure `Extract-Cores` then terminates, after which `IHS` updates $ub = \text{COST}(\mathcal{F}, \tau) = 2$. Since $ub = lb$, the algorithm terminates and returns τ as an optimal solution of $\mathcal{F}$.

目录

In theory

Observations:

- If we add a new hard clause, a new soft literal, or change the weight of a soft literal, all extracted cores are still valid
  - cores can be preserved between solver invocations
  - only objective needs to be altered in the IP solver
- The SAT solver knows nothing about the weights of soft literals
  - add hard clauses directly to the SAT solver
  - no need to reinitialize

How to deal with assumptions without restarting the SAT solver?

In theory

Observations:

- If we add a new hard clause, a new soft literal, or change the weight of a soft literal, all extracted cores are still valid
  - cores can be preserved between solver invocations
  - only objective needs to be altered in the IP solver
- The SAT solver knows nothing about the weights of soft literals
  - add hard clauses directly to the SAT solver
  - no need to reinitialize

How to deal with assumptions without restarting the SAT solver?

In theory

Main idea: pass user-provided assumptions A along with IHS solving assumptions $\neg(\mathcal{F}_L \backslash hs)$ to the internal SAT solver

- if $a \in A \cap \mathcal{F}_L$, do not include $\neg a$ as assumption from $\neg(\mathcal{F}_L \backslash hs)$
- cores extracted during search may also contain literals from $\neg A$
  - How to preserve cores when solving under assumptions?

Conditional Cores

Given a MaxSAT instance $(\mathcal{F}_H, S, w)$, a conditional core with respect to assumptions A is a clause $\kappa^a \subset \neg A \cup S$ that is entailed by F_H. The restriction of a conditional core is $\kappa^a \backslash \neg A$.

In theory

Main idea: pass user-provided assumptions A along with IHS solving assumptions $\neg(\mathcal{F}_L \backslash hs)$ to the internal SAT solver

- if $a \in A \cap \mathcal{F}_L$, do not include $\neg a$ as assumption from $\neg(\mathcal{F}_L \backslash hs)$
- cores extracted during search may also contain literals from $\neg A$
  - How to preserve cores when solving under assumptions?

Conditional Cores

Given a MaxSAT instance $(\mathcal{F}_H, S, w)$, a conditional core with respect to assumptions A is a clause $\kappa^a \subset \neg A \cup S$ that is entailed by F_H. The restriction of a conditional core is $\kappa^a \backslash \neg A$.

Algorithm 2 Incremental MaxSAT solving under assumptions.

IHS-assumptions($\mathcal{F}$, *cond*-$\mathcal{C}$, A)

Input: An instance $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_L, w)$, a set A of assumptions, a set cond-$\mathcal{C}$ of conditional cores

Output: An optimal solution τ under the assumptions A

$lb \leftarrow 0$; $ub \leftarrow \infty$; $\tau_{best} \leftarrow \varnothing$; $\mathcal{C} \leftarrow \emptyset$;

for $\kappa^a \in$ *cond*-$\mathcal{C}$ **do**

  $\kappa \leftarrow \kappa^a \setminus \neg A$;

  if $\kappa^a \cap A = \emptyset \wedge (\kappa \subset \mathcal{F}_L)$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{\kappa\}$;

while *(TRUE)* **do**

  $hs \leftarrow$ Min-Hs($\mathcal{F}_L$,$\mathcal{C}$);

  $lb = \textsc{cost}(\mathcal{F}, hs)$;

  if $(lb = ub)$ **then break**;

  $(K, \tau) \leftarrow$ Extract-Cores-Assumptions($\mathcal{F}_H, \mathcal{F}_L, A, hs$);

  if $(\textsc{cost}(\mathcal{F}, \tau) < ub)$ **then** $\tau_{best} \leftarrow \tau$; $ub \leftarrow \textsc{cost}(\mathcal{F}, \tau)$;

  if $(lb = ub)$ **then return** τ_{best};

  for $\kappa^a \in K$ **do**

    cond-$\mathcal{C} \leftarrow$ cond-$\mathcal{C} \cup \{\kappa^a\}$;

    $\mathcal{C} \leftarrow \mathcal{C} \cup \{\kappa^a \setminus \neg A\}$;

In theory

With current MaxSAT assumptions A :

- Include A in the assumptions of every internal SAT solver call (and remove conflicting soft literals)
  - models reported by the SAT solver will satisfy A
- SAT solver extracts conditional cores κ^a
  - add κ^a to a set of all collected conditional cores
  - add the restriction $\kappa^a \setminus \neg A$ to the IP solver

With next MaxSAT assumptions A' :

- Reinitialize the IP solver
- Check all known conditional cores κ^a
  - if $\kappa^a \cap A' = \emptyset$ and the restriction $\kappa^a \setminus \neg A' \subseteq S$, add the restriction to the IP solver

No need to reinitialize the SAT solver, and cores are preserved.

In theory

With current MaxSAT assumptions A :

- Include A in the assumptions of every internal SAT solver call (and remove conflicting soft literals)
  - models reported by the SAT solver will satisfy A
- SAT solver extracts conditional cores κ^a
  - add κ^a to a set of all collected conditional cores
  - add the restriction $\kappa^a \setminus \neg A$ to the IP solver

With next MaxSAT assumptions A' :

- Reinitialize the IP solver
- Check all known conditional cores κ^a
  - if $\kappa^a \cap A' = \emptyset$ and the restriction $\kappa^a \setminus \neg A' \subseteq S$, add the restriction to the IP solver

No need to reinitialize the SAT solver, and cores are preserved.

In theory

With current MaxSAT assumptions A :

- Include A in the assumptions of every internal SAT solver call (and remove conflicting soft literals)
  - models reported by the SAT solver will satisfy A
- SAT solver extracts conditional cores κ^a
  - add κ^a to a set of all collected conditional cores
  - add the restriction $\kappa^a \setminus \neg A$ to the IP solver

With next MaxSAT assumptions A' :

- Reinitialize the IP solver
- Check all known conditional cores κ^a
  - if $\kappa^a \cap A' = \emptyset$ and the restriction $\kappa^a \setminus \neg A' \subseteq S$, add the restriction to the IP solver

No need to reinitialize the SAT solver, and cores are preserved.

In theory

With current MaxSAT assumptions A :

- Include A in the assumptions of every internal SAT solver call (and remove conflicting soft literals)
  - models reported by the SAT solver will satisfy A
- SAT solver extracts conditional cores κ^a
  - add κ^a to a set of all collected conditional cores
  - add the restriction $\kappa^a \setminus \neg A$ to the IP solver

With next MaxSAT assumptions A' :

- Reinitialize the IP solver
- Check all known conditional cores κ^a
  - if $\kappa^a \cap A' = \emptyset$ and the restriction $\kappa^a \setminus \neg A' \subseteq S$, add the restriction to the IP solver

No need to reinitialize the SAT solver, and cores are preserved.

In theory

With current MaxSAT assumptions A :

- Include A in the assumptions of every internal SAT solver call (and remove conflicting soft literals)
  - models reported by the SAT solver will satisfy A
- SAT solver extracts conditional cores κ^a
  - add κ^a to a set of all collected conditional cores
  - add the restriction $\kappa^a \setminus \neg A$ to the IP solver

With next MaxSAT assumptions A' :

- Reinitialize the IP solver
- Check all known conditional cores κ^a
  - if $\kappa^a \cap A' = \emptyset$ and the restriction $\kappa^a \setminus \neg A' \subseteq S$, add the restriction to the IP solver

No need to reinitialize the SAT solver, and cores are preserved.

Example 1

- **Example 1.** Consider the MaxSAT instance $\mathcal{F} = (\mathcal{F}_H, \mathcal{F}_L, w)$ with $\mathcal{F}_H = \{(b_1 \vee x), (\neg x \vee b_2), (\neg z), (z \vee y \vee b_3 \vee b_4), (\neg y \vee b_3 \vee b_4)\}$, $\mathcal{F}_L = \{b_1, b_2, b_3, b_4\}$ and $w(b_1) = w(b_3) = w(b_4) = 1$ and $w(b_2) = 2$. The solution $\tau = \{b_1, \neg b_2, b_3, \neg b_4, \neg x, y, \neg z\}$ is an optimal solution of $\mathcal{F}$, with $\textsc{cost}(\mathcal{F}, \tau) = \textsc{cost}(\mathcal{F}) = 2$.

Example 4

▶ **Example 4.** Consider an invocation of `IHS-assumptions` on the MaxSAT instance $\mathcal{F}$ from Example 1 under the assumptions $A = \{x\}$. For clarity, here we will ignore the set `cond-`$\mathcal{C}$. Initially $\mathcal{C} = \emptyset$, so the first hitting set $hs = \emptyset$, As such $lb = \textsc{cost}(\mathcal{F}, hs) = 0 < \infty = ub$, so the algorithm invokes `Extract-Cores-Assumptions`$(\mathcal{F}_H, \mathcal{F}_L, \{x\}, \emptyset)$. The procedure extracts conditional cores of $\mathcal{F}$ by invoking a SAT solver under the assumptions `Solver-A` $= \{\neg b \mid b \in \mathcal{F}_L \setminus hs\} \cup A = \{\neg b_1, \neg b_2, \neg b_3, \neg b_4, x\}$. The result is "unsatisfiable". Assume the solver returns the conditional core $\kappa^a = \{\neg x, b_2\}$. The next solver call is made under the assumptions `Solver-A` $= \{\neg b_1, \neg b_3, \neg b_4, x\}$. The result is again "unsatisfiable" and the solver returns the (conditional) core $\{b_3, b_4\}$. The third solver call returns "satisfiable" and (for example) the solution $\tau = \{\neg b_1, b_2, \neg b_3, b_4, x, y, \neg z\}$ so `Extract-Cores-Assumptions` terminates. The upper bound is updated by $ub = \textsc{cost}(\mathcal{F}, \tau) = 3$ and the restrictions of each conditional core added to $\mathcal{C}$. In the next iteration $\mathcal{C} = \{\{b_2\}, \{b_3, b_4\}\}$ and the `Min-Hs` procedure returns (for example) $hs = \{b_2, b_3\}$. This hitting set updates the lower bound to $lb = \textsc{cost}(\mathcal{F}, hs) = 3 = ub$ so the algorithm terminates and returns τ as an optimal solution to $\mathcal{F}$ under A.

In practice

Make use of **MaxHS: state-of-the-art IHS-based MaxSAT solver**. Realizing incrementality requires a non-trivial amount of engineering.

- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores. To extract valid cores, perform unit propagation under current MaxSAT assumptions.
  - removes redundant cores and simplifies them
  - still need to check that the resulting cores only contain soft literals
- **IPAMIR wrapper:** When initialized, MaxHS performs several rounds of simplification to the input formula.
  - variable mappings must be maintained

In practice

Make use of **MaxHS: state-of-the-art IHS-based MaxSAT solver**. Realizing incrementality requires a non-trivial amount of engineering.

- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores. To extract valid cores, perform unit propagation under current MaxSAT assumptions.
  - removes redundant cores and simplifies them
  - still need to check that the resulting cores only contain soft literals
- **IPAMIR wrapper:** When initialized, MaxHS performs several rounds of simplification to the input formula.
  - variable mappings must be maintained

目录

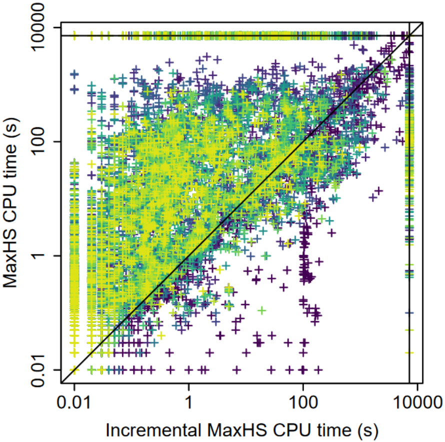
Empirical Evaluation

- Benchmark instances
  - All 1184 instances from complete tracks of MaxSAT Evaluation 2021
  - For each benchmark, create 20 different sets of assumptions by hardening each soft clause with probability 0.01.(23680 iterations overall)
- Benchmark setup
  - iMaxHS vs. its non-incremental version in default settings
    - for non-incremental, add assumptions directly as hard clauses
  - Per-instance limits: 7200 seconds and 16 GB memory
    - instance: 20 MaxSAT solver calls each with different assumptions
    - exclude WCNF parsing times from consideration

Empirical Evaluation

- Benchmark instances
  - All 1184 instances from complete tracks of MaxSAT Evaluation 2021
  - For each benchmark, create 20 different sets of assumptions by hardening each soft clause with probability 0.01.(23680 iterations overall)
- Benchmark setup
  - iMaxHS vs. its non-incremental version in default settings
    - for non-incremental, add assumptions directly as hard clauses
  - Per-instance limits: 7200 seconds and 16 GB memory
    - instance: 20 MaxSAT solver calls each with different assumptions
    - exclude WCNF parsing times from consideration

Result


MaxHS CPU time (s)
0.01
1
100
10000
Incremental MaxHS CPU time (s)
0.01
1
100
10000

Thank you