Incremental Maximum Satisfiability

Andreas Niskanen   Jeremias Berg   Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland

July 6 @ EURO 2022, Espoo, Finland

Maximum satisfiability (MaxSAT)

Bacchus, Järvisalo, and Martins [2021]

- Optimization paradigm based on Boolean satisfiability (SAT)
  - *minimize:* linear objective function over 0-1 variables
  - *subject to:* constraints expressed in propositional logic
- Suitable **declarative modelling language** for various real-world optimization problems involving **logical constraints**
  - verification of hardware and software
  - planning and scheduling
  - interpretable machine learning
  - ...
- Significant **progress in solving technology** over the past 20 years
  - multiple different native solving algorithms
  - state-of-the-art solvers build on the success of SAT solvers

Maximum satisfiability (MaxSAT)

Bacchus, Järvisalo, and Martins [2021]

- Optimization paradigm based on Boolean satisfiability (SAT)
  - *minimize:* linear objective function over 0-1 variables
  - *subject to:* constraints expressed in propositional logic
- Suitable **declarative modelling language** for various real-world optimization problems involving **logical constraints**
  - verification of hardware and software
  - planning and scheduling
  - interpretable machine learning
  - ...
- Significant **progress in solving technology** over the past 20 years
  - multiple different native solving algorithms
  - state-of-the-art solvers build on the success of SAT solvers

Maximum satisfiability (MaxSAT)

Bacchus, Järvisalo, and Martins [2021]

- Optimization paradigm based on Boolean satisfiability (SAT)
  - *minimize:* linear objective function over 0-1 variables
  - *subject to:* constraints expressed in propositional logic
- Suitable **declarative modelling language** for various real-world optimization problems involving **logical constraints**
  - verification of hardware and software
  - planning and scheduling
  - interpretable machine learning
  - ...
- Significant **progress in solving technology** over the past 20 years
  - multiple different native solving algorithms
  - state-of-the-art solvers build on the success of SAT solvers

Incremental optimization *(a.k.a. reoptimization)*

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - types of incremental changes applied between instances:
    - 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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers
- Application scenarios for incremental MaxSAT known, but...
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental optimization *(a.k.a. reoptimization)*

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - types of incremental changes applied between instances:
    - 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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers
- Application scenarios for incremental MaxSAT known, but...
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental optimization *(a.k.a. reoptimization)*

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - types of incremental changes applied between instances:
    - 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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers
- Application scenarios for incremental MaxSAT known, but...
- Currently **MaxSAT solvers offer limited support** for incrementality

Incremental optimization *(a.k.a. reoptimization)*

- Various problem domains call for iterative solving procedures where **a sequence of related instances are solved**
  - types of incremental changes applied between instances:
    - 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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers
- Application scenarios for incremental MaxSAT known, but...
- Currently **MaxSAT solvers offer limited support** for incrementality

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

1. Detail **various forms of incrementality in MaxSAT**
   - adding constraints, changing objective, 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**

Implementation and benchmark data openly available:
`https://bitbucket.org/coreo-group/incremental-maxhs`

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

1. Detail **various forms of incrementality in MaxSAT**
   - adding constraints, changing objective, 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**

Implementation and benchmark data openly available:
`https://bitbucket.org/coreo-group/incremental-maxhs`

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

1. Detail **various forms of incrementality in MaxSAT**
   - adding constraints, changing objective, 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**

Implementation and benchmark data openly available:
https://bitbucket.org/coreo-group/incremental-maxhs

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

1. Detail **various forms of incrementality in MaxSAT**
   - adding constraints, changing objective, 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**

Implementation and benchmark data openly available:
`https://bitbucket.org/coreo-group/incremental-maxhs`

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

1. Detail **various forms of incrementality in MaxSAT**
   - adding constraints, changing objective, 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**

Implementation and benchmark data openly available:
`https://bitbucket.org/coreo-group/incremental-maxhs`

Maximum satisfiability (MaxSAT)

- Optimization extension of Boolean Satisfiability (SAT)
  - reasoning over logical constraints: and, or, exclusive-or, if-then
- Boolean (0-1) optimization paradigm
  - hard constraints encoded using *clauses* i.e. logical ORs: specific type of at-least constraints
  - linear objective function
- Goal: find an assignment which satisfies all hard constraints and minimizes the objective function

Maximum satisfiability (MaxSAT)

- Optimization extension of Boolean Satisfiability (SAT)
  - reasoning over logical constraints: and, or, exclusive-or, if-then
- Boolean (0-1) optimization paradigm
  - hard constraints encoded using *clauses* i.e. logical ORs: specific type of at-least constraints
  - linear objective function
- Goal: find an assignment which satisfies all hard constraints and minimizes the objective function

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Optimal solution:
$x = 1,\ y = 0,\ z = 0$

Incremental MaxSAT

- Aim for solving a sequence of related MaxSAT instances efficiently, avoiding computation from scratch
- Different scenarios call for different forms of incremental changes
  - adding or removing hard constraints
  - modifying the objective function
  - solving under assumptions: partial assignments to variables

Adding hard constraints

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Check whether it satisfies a desired property:
  if not, exclude it (and other non-solutions) from consideration

Generic paradigm with various instantiations employing MaxSAT

Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Adding hard constraints

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Check whether it satisfies a desired property:
  if not, exclude it (and other non-solutions) from consideration

Generic paradigm with various instantiations employing MaxSAT

Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Adding hard constraints

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Check whether it satisfies a desired property:
  if not, exclude it (and other non-solutions) from consideration

Generic paradigm with various instantiations employing MaxSAT

Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Optimal solution:
$x = 1,\ y = 0,\ z = 0$

Adding hard constraints

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Check whether it satisfies a desired property:
  if not, exclude it (and other non-solutions) from consideration

Generic paradigm with various instantiations employing MaxSAT

Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \\ & (1 - x) + y + z \geq 1 \end{aligned}$$

~~Optimal solution~~:
$x = 1,\ y = 0,\ z = 0$

Adding hard constraints

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Check whether it satisfies a desired property:
  if not, exclude it (and other non-solutions) from consideration

Generic paradigm with various instantiations employing MaxSAT

Mangal, Zhang, Nori, and Naik [2015]; Niskanen and Järvisalo [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \\ & (1 - x) + y + z \geq 1 \end{aligned}$$

Optimal solution:
$x = 0,\ y = 1,\ z = 1$

Changing coefficients

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Give more priority to more diverse solutions and repeat

Learning classifiers with the AdaBoost algorithm: MaxSAT employed for decision trees Hu, Siala, Hebrard, and Huguet [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Changing coefficients

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Give more priority to more diverse solutions and repeat

Learning classifiers with the AdaBoost algorithm:
MaxSAT employed for decision trees *Hu, Siala, Hebrard, and Huguet [2020]*

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Changing coefficients

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Give more priority to more diverse solutions and repeat

Learning classifiers with the AdaBoost algorithm:
MaxSAT employed for decision trees *Hu, Siala, Hebrard, and Huguet [2020]*

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Optimal solution:
$x = 1,\ y = 0,\ z = 0$

Changing coefficients

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Give more priority to more diverse solutions and repeat

Learning classifiers with the AdaBoost algorithm:
MaxSAT employed for decision trees — *Hu, Siala, Hebrard, and Huguet [2020]*

$$\begin{aligned} \text{minimize:} \quad & 3x + y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

~~Optimal~~ solution:
$x = 1,\ y = 0,\ z = 0$

Changing coefficients

Example

Consider an initial problem instance, and an iterative procedure:

- Compute an optimal solution to the current instance
- Give more priority to more diverse solutions and repeat

Learning classifiers with the AdaBoost algorithm:
MaxSAT employed for decision trees *Hu, Siala, Hebrard, and Huguet [2020]*

$$\begin{aligned} \text{minimize:} \quad & 3x + y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Optimal solution:
$x = 0,\ y = 1,\ z = 1$

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$
\begin{aligned}
\text{minimize:} \quad & x + 2y \\
\text{subject to:} \quad & x + y \geq 1 \\
& y + (1 - z) \geq 1
\end{aligned}
$$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \end{aligned}$$

Optimal solution:
$x = 1$, $y = 0$, $z = 0$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \\ \text{assuming:} \quad & z = 1 \end{aligned}$$

~~Optimal solution~~:
$x = 1,\ y = 0,\ z = 0$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \\ \text{assuming:} \quad & z = 1 \end{aligned}$$

Optimal solution:
$x = 0,\ y = 1,\ z = 1$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

Optimizing under assumptions

Example

Consider an initial problem instance, and an iterative procedure:

- Extract information about the current state of the world
- Incorporate it to the instance and compute an optimal solution

Timetabling under disruptions: time or room slots may become unavailable

Lemos, Monteiro, and Lynce [2020]

$$\begin{aligned} \text{minimize:} \quad & x + 2y \\ \text{subject to:} \quad & x + y \geq 1 \\ & y + (1 - z) \geq 1 \\ \text{assuming:} \quad & z = 1 \end{aligned}$$

Optimal solution:

$x = 0,\ y = 1,\ z = 1$

- Unlike hard constraints, assumptions are revertable
  - removal of hard constraints can be simulated with assumptions

IPAMIR: Incremental API for MaxSAT

- Generic interface for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on IPASIR: standard interfaca for incremental SAT
- Specifies incremental changes to a MaxSAT instance
  - adding hard constraints
  - adding terms to or changing coefficients of the objective function
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Incremental API for MaxSAT

- Generic interface for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on IPASIR: standard interfaca for incremental SAT
- Specifies incremental changes to a MaxSAT instance
  - adding hard constraints
  - adding terms to or changing coefficients of the objective function
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Incremental API for MaxSAT

- Generic interface for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on IPASIR: standard interfaca for incremental SAT
- Specifies incremental changes to a MaxSAT instance
  - adding hard constraints
  - adding terms to or changing coefficients of the objective function
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Incremental API for MaxSAT

- Generic interface for incremental MaxSAT
  - for MaxSAT solvers providing support for incrementality
  - for applications making use of incrementality
- Builds on IPASIR: standard interfaca for incremental SAT
- Specifies incremental changes to a MaxSAT instance
  - adding hard constraints
  - adding terms to or changing coefficients of the objective function
  - assumptions on variables
- Includes other essential declarations
  - constructing and releasing a solver
  - solving, variable assignments, objective values

IPAMIR: Incremental API for MaxSAT

```
// 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));
```

Interface and example applications openly available:
https://bitbucket.org/coreo-group/ipamir

Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]

An iterative approach: identify *sources of inconsistency* and *repair the inconsistencies* in a minimal way.

- Central notion: a *core* is an assignment to a subset of the objective function which cannot be extended to satisfy the hard constraints
  - 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 coefficients of the objective
  - 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

Davies and Bacchus [2011, 2013]

An iterative approach: identify *sources of inconsistency*
and *repair the inconsistencies* in a minimal way.

- Central notion: a *core* is an assignment to a subset of the objective function which cannot be extended to satisfy the hard constraints
  - 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 coefficients of the objective
  - 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

Davies and Bacchus [2011, 2013]

An iterative approach: identify *sources of inconsistency* and *repair the inconsistencies* in a minimal way.

- Central notion: a *core* is an assignment to a subset of the objective function which cannot be extended to satisfy the hard constraints
  - 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 coefficients of the objective
  - 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

Davies and Bacchus [2011, 2013]

An iterative approach: identify *sources of inconsistency* and *repair the inconsistencies* in a minimal way.

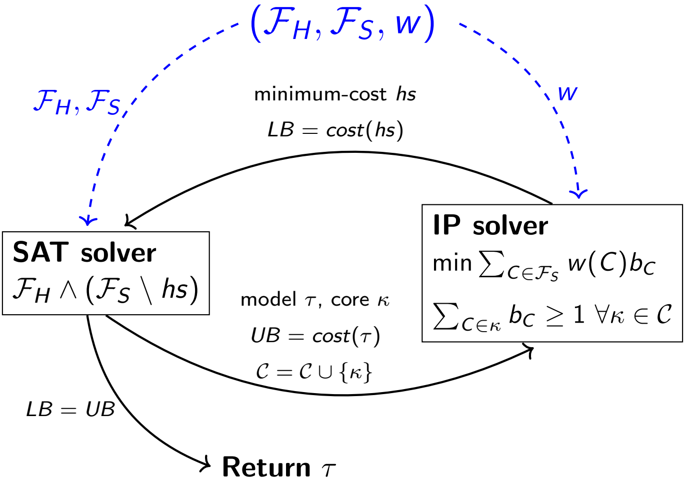
- Central notion: a *core* is an assignment to a subset of the objective function which cannot be extended to satisfy the hard constraints
  - SAT solver as *core extractor*
- *hs* is a *hitting set* over a set of cores $\mathcal{C}$ if *hs* intersects each $\kappa \in \mathcal{C}$
  - *cost* of a hitting set determined by coefficients of the objective
  - 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

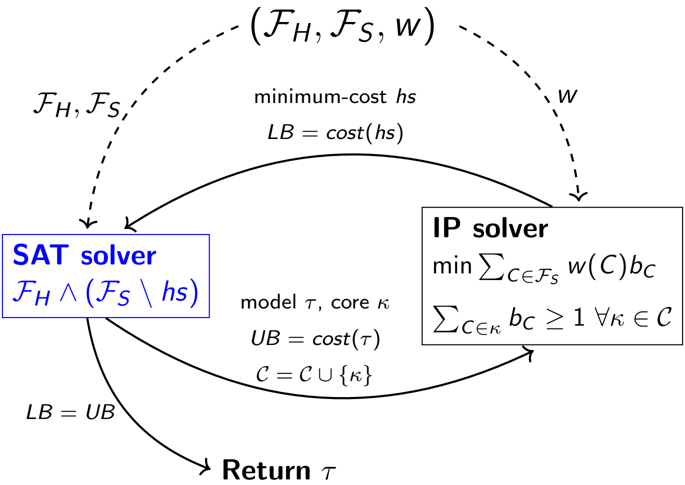
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



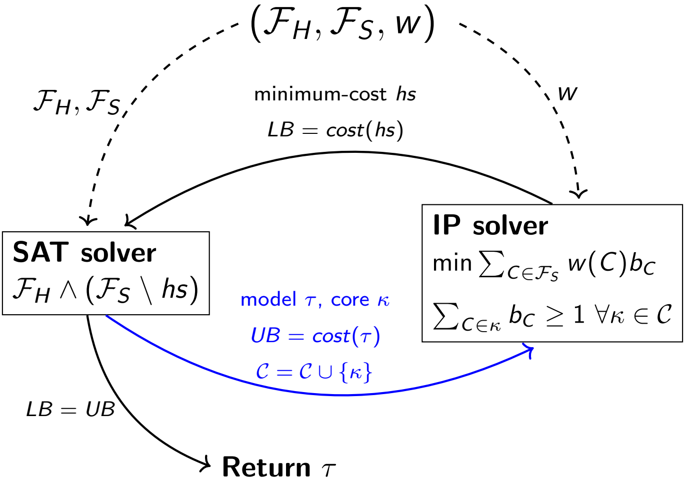
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



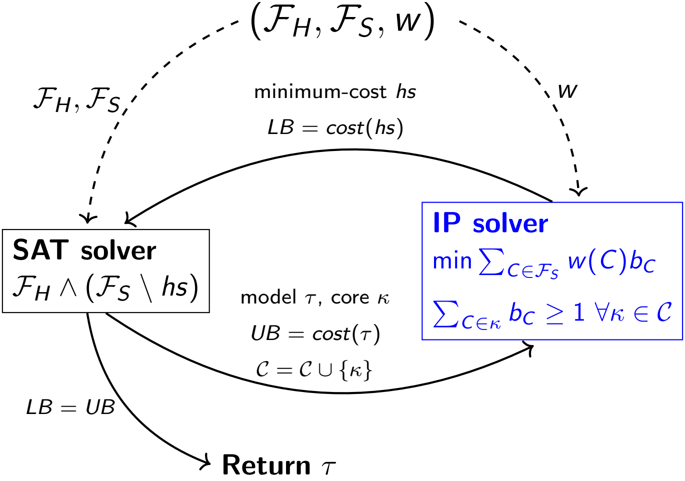
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



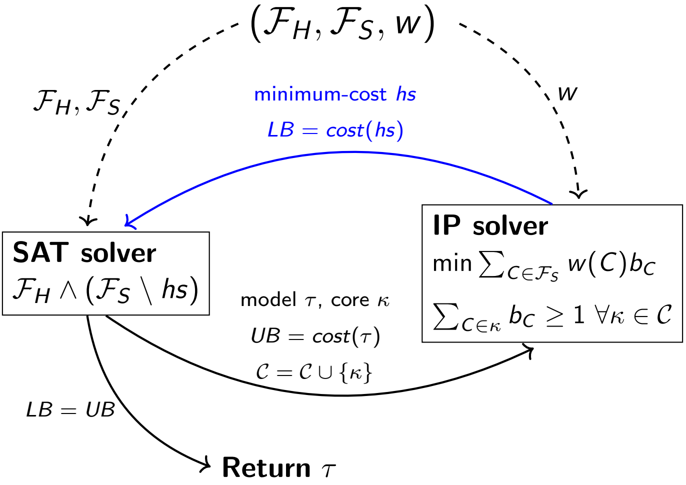
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



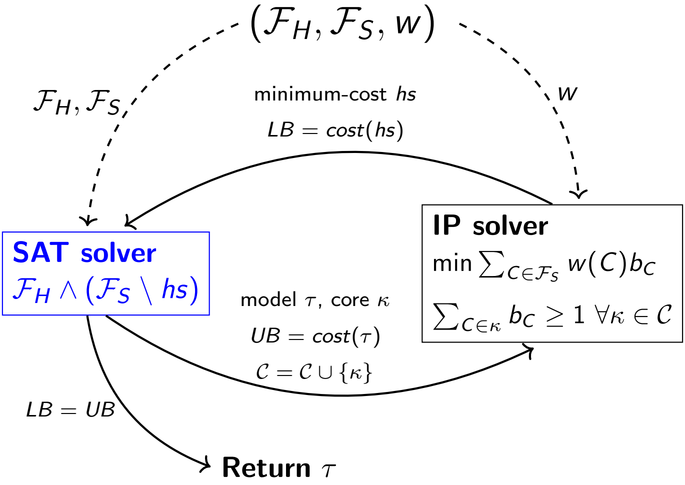
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



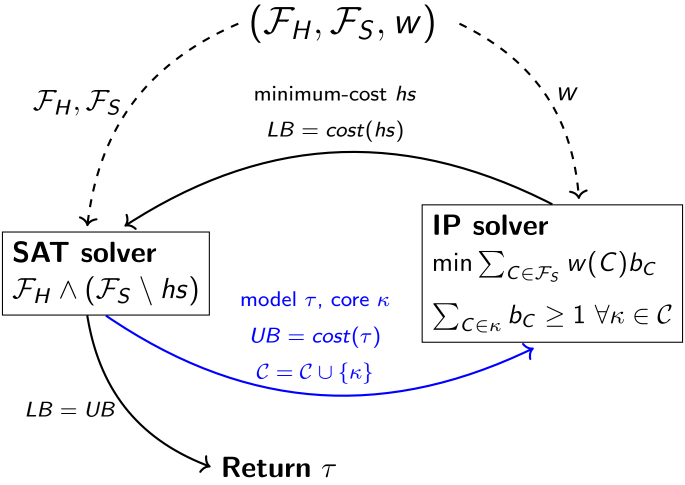
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



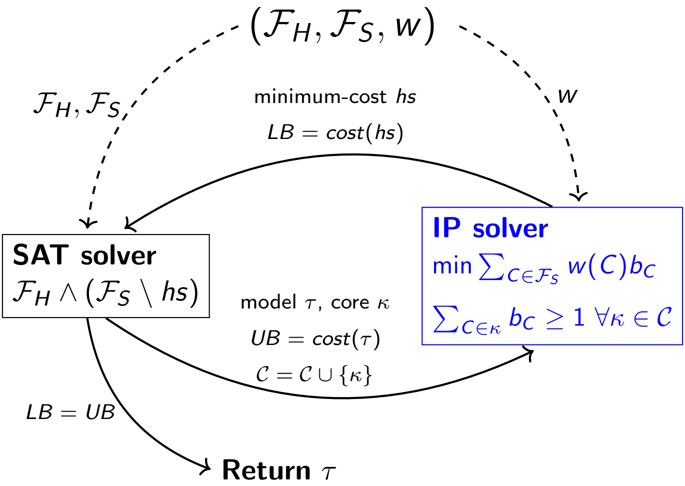
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



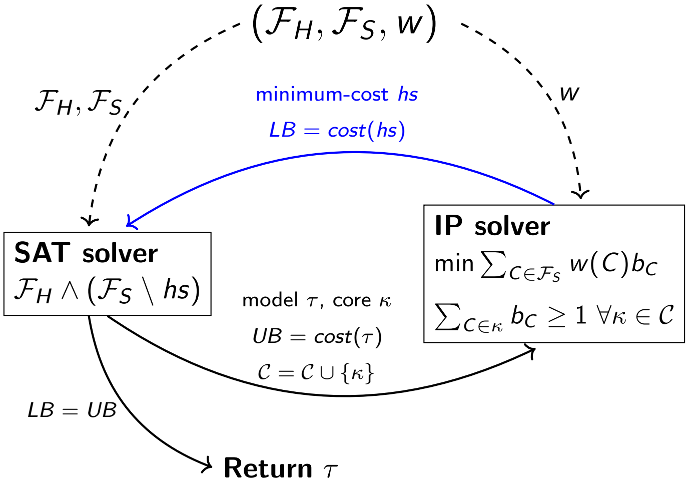
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



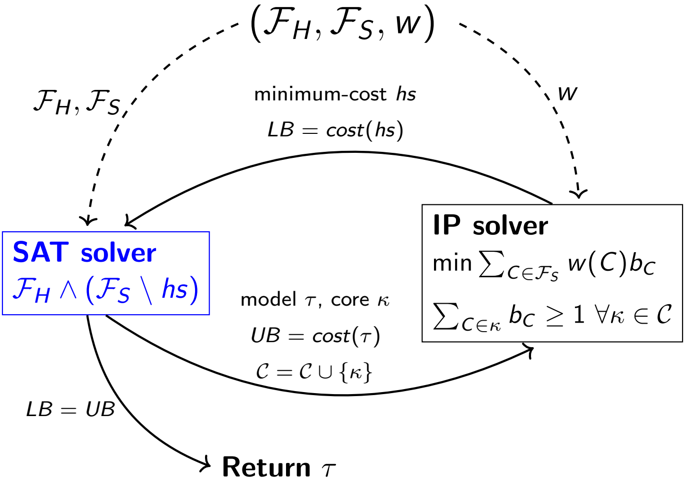
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



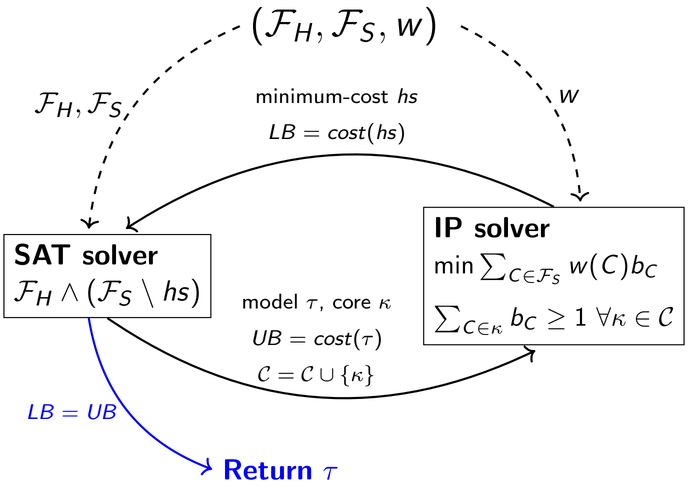
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



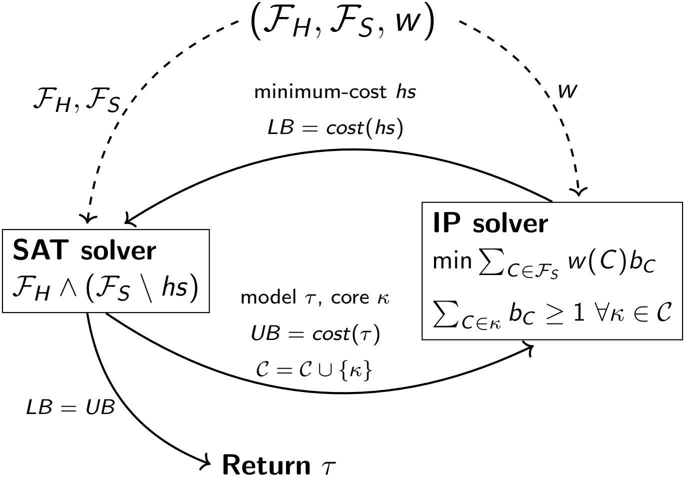
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



Incremental IHS

In theory

Observations:

- If we add a new hard clause, a term to the objective function, or change coefficients, **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 objective
    - add hard clauses directly to the SAT solver
    - no need to reinitialize

Assumptions require more care: the notion of *conditional cores* take into account the assumptions made during core extraction

- still, **no need to reset the SAT solver**
- however, IP solver reinitialized with *restrictions* of all conditional cores that are valid under current assumptions

Incremental IHS

In theory

Observations:

- If we add a new hard clause, a term to the objective function, or change coefficients, **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 objective
  - add hard clauses directly to the SAT solver
  - no need to reinitialize

Assumptions require more care: the notion of *conditional cores* take into account the assumptions made during core extraction

- still, **no need to reset the SAT solver**
- however, IP solver reinitialized with *restrictions* of all conditional cores that are valid under current assumptions

Incremental IHS

In practice

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

- **Simplification:** when initialized, MaxHS performs several rounds of simplification to the input MaxSAT instance
  - variable mappings between internal and external representations
  - fixed variables need to be handled correctly
  - ...
- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores
  - removes redundant cores and simplifies them
- **Other techniques:** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

Incremental IHS

In practice

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

- **Simplification:** when initialized, MaxHS performs several rounds of simplification to the input MaxSAT instance
  - variable mappings between internal and external representations
  - fixed variables need to be handled correctly
  - ...
- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores
  - removes redundant cores and simplifies them
- **Other techniques:** must be modified to preserve correctness
  - reduced cost fixing Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores Berg, Bacchus, and Poole [2020]

Incremental IHS

In practice

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

- **Simplification:** when initialized, MaxHS performs several rounds of simplification to the input MaxSAT instance
    - variable mappings between internal and external representations
    - fixed variables need to be handled correctly
    - ...
- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores
    - removes redundant cores and simplifies them
- **Other techniques:** must be modified to preserve correctness
    - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
    - abstract cores — Berg, Bacchus, and Poole [2020]

Incremental IHS

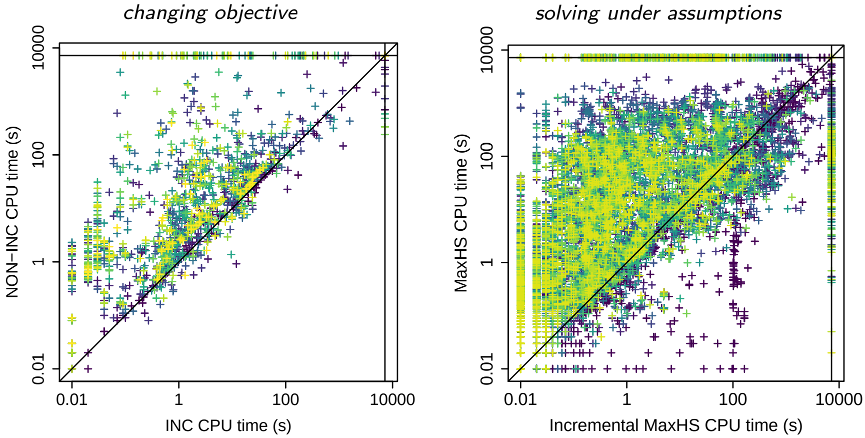
In practice

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

- **Simplification:** when initialized, MaxHS performs several rounds of simplification to the input MaxSAT instance
  - variable mappings between internal and external representations
  - fixed variables need to be handled correctly
  - ...
- **Maintaining conditional cores:** use another SAT solver as a database for storing conditional cores
  - removes redundant cores and simplifies them
- **Other techniques:** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

Empirical evaluation

Increased performance gained by preserving cores:



- blue points → earlier iterations
- yellow points → later iterations

Summary

Contributions

- **IPAMIR:** incremental API for MaxSAT
  - details various forms of incrementality in MaxSAT
  - provides a standard interface to facilitate the development of solvers and applications
- **Incremental MaxHS:** fully-fledged incremental MaxSAT solver
  - supports all IPAMIR functionality
  - preserves cores and does not reset SAT solver between invocations
- **Empirical evaluation:** clear benefit from incrementality

Implementation available online in open source:
`https://bitbucket.org/coreo-group/incremental-maxhs`

Thank you for your attention!

Get in touch via email:
`andreas.niskanen@helsinki.fi`

Or come chat in person :)

Bibliography I

Fahiem Bacchus, Antti Hyttinen, Matti Järvisalo, and Paul Saikko. Reduced cost fixing in MaxSAT. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, volume 10416 of *Lecture Notes in Computer Science*, pages 641–651. Springer, 2017. doi: 10.1007/978-3-319-66158-2_41.

Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability, Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, chapter 24, pages 929–991. IOS Press, 2021. doi: 10.3233/FAIA201008.

Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set MaxSat solving. In Luca Pulina and Martina Seidl, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 277–294. Springer, 2020. doi: 10.1007/978-3-030-51825-7_20.

Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In Jimmy Ho-Man Lee, editor, *Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011, Proceedings*, volume 6876 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2011. doi: 10.1007/978-3-642-23786-7_19.

Jessica Davies and Fahiem Bacchus. Postponing optimization to speed up MAXSAT solving. In Christian Schulte, editor, *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013, Proceedings*, volume 8124 of *Lecture Notes in Computer Science*, pages 247–262. Springer, 2013. doi: 10.1007/978-3-642-40627-0_21.

Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003. doi: 10.1016/S1571-0661(05)82542-3.

Hao Hu, Mohamed Siala, Emmanuel Hebrard, and Marie-José Huguet. Learning optimal decision trees with MaxSAT and its integration in AdaBoost. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1170–1176. ijcai.org, 2020. doi: 10.24963/ijcai.2020/163.

Bibliography II

Alexandre Lemos, Pedro T. Monteiro, and Inês Lynce. Minimal perturbation in university timetabling with maximum satisfiability. In Emmanuel Hebrard and Nysret Musliu, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 17th International Conference, CPAIOR 2020, Vienna, Austria, September 21-24, 2020, Proceedings*, volume 12296 of *Lecture Notes in Computer Science*, pages 317–333. Springer, 2020. doi: 10.1007/978-3-030-58942-4_21.

Ravi Mangal, Xin Zhang, Aditya V. Nori, and Mayur Naik. Volt: A lazy grounding framework for solving very large maxsat instances. In Marijn Heule and Sean A. Weaver, editors, *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, volume 9340 of *Lecture Notes in Computer Science*, pages 299–306. Springer, 2015. doi: 10.1007/978-3-319-24318-4_22.

Andreas Niskanen and Matti Järvisalo. Strong refinements for hard problems in argumentation dynamics. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 841–848. IOS Press, 2020. doi: 10.3233/FAIA200174.

Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Enabling incrementality in the implicit hitting set approach to maxsat under changing weights. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.CP.2021.44.

Andreas Niskanen, Jeremias Berg, and Matti Järvisalo. Incremental maximum satisfiability. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, Haifa, Israel, August 2-5, 2022*, volume 236 of *LIPIcs*, pages 14:1–14:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. To appear.