Incremental Maximum Satisfiability

Andreas Niskanen Jeremias Berg Matti Järvisalo

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

August 2 @ SAT 2022 @ FLoC 2022, Haifa, Israel

Maximum satisfiability (MaxSAT)

- Declarative **optimization paradigm** based on SAT
  - hard constraints as propositional clauses
  - minimize linear objective function
- Suitable **declarative modelling language** for various real-world optimization problems
  - planning, scheduling, verification, data analysis, machine learning, knowledge representation and reasoning, ...
- State-of-the-art solvers build on the success of SAT solvers
  - significant progress in MaxSAT solver technology
  - incremental API for SAT essential in practical implementations

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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality
  - despite potentially useful information that could be preserved across solver invocations: state of SAT solver, cores

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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality
  - despite potentially useful information that could be preserved across solver invocations: state of SAT solver, cores

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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality
  - despite potentially useful information that could be preserved across solver invocations: state of SAT solver, cores

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 Eén and Sörensson [2003]
  - extensively applied by MaxSAT solvers, QBF solvers, etc.
- Currently **MaxSAT solvers offer limited support** for incrementality
  - despite potentially useful information that could be preserved across solver invocations: state of SAT solver, cores

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

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

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 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

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 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

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 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

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

Contributions

Niskanen, Berg, and Järvisalo [2021, 2022]

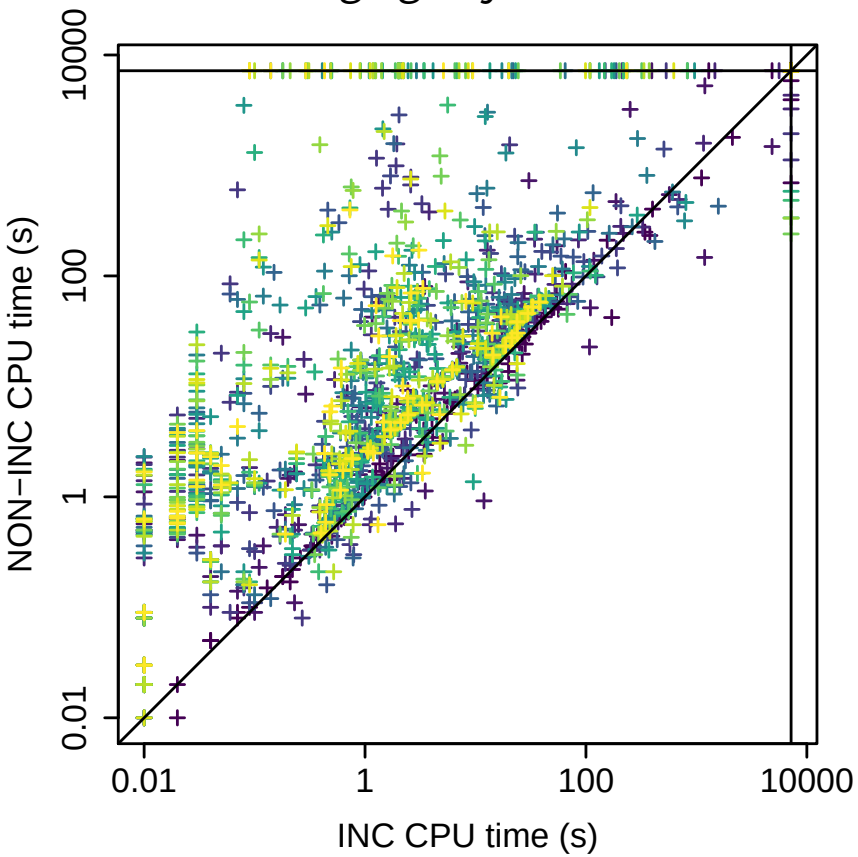
1. Detail **various forms of incrementality in MaxSAT**
   - adding hard clauses, soft literals, assumptions
2. Propose **IPAMIR: incremental API for MaxSAT**
   - generic interface for developing incremental MaxSAT solvers and applications making use of incremental MaxSAT
   - MaxSAT Evaluation 2022: 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

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

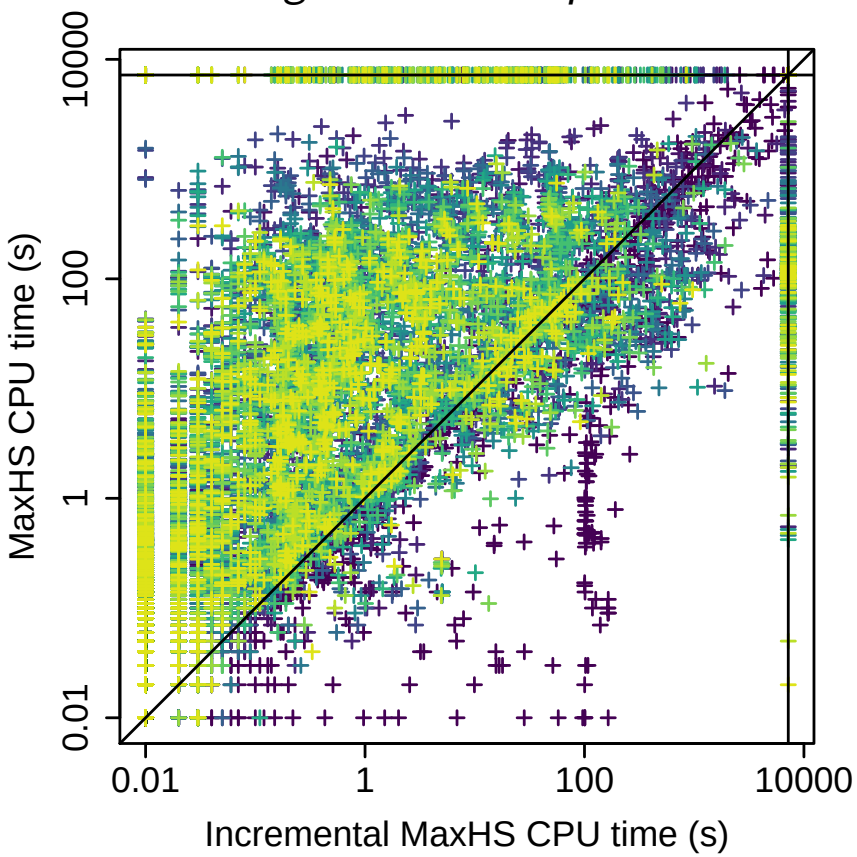
iMaxHS

An extension of the IHS solver MaxHS that supports incrementality

changing objective



solving under assumptions



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

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 $\sum_{b \in S} w(b) \cdot b$.

Note: definition equivalent to weighted soft clauses $\mathcal{F}_S$:

- *relax each soft clause $C \in \mathcal{F}_S$ to $C \vee \neg b_C$,*
- *add $C \vee \neg b_C$ to $\mathcal{F}_H$, and b_C to S with weight $w(b_C) = w(C)$.*

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 $\sum_{b \in S} w(b) \cdot b$.

Note: definition equivalent to weighted soft clauses $\mathcal{F}_S$:

- *relax each soft clause $C \in \mathcal{F}_S$ to $C \vee \neg b_C$,*
- *add $C \vee \neg b_C$ to $\mathcal{F}_H$, and b_C to S with weight $w(b_C) = w(C)$.*

Incremental changes in 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 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]

Note: similarly as in SAT, assumptions can be used to simulate the removal of clauses.

Incremental changes in 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 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]

Note: similarly as in SAT, assumptions can be used to simulate the removal of clauses.

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 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: 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 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: 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 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: 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 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));
```

Functions declared in the IPAMIR header.

In contrast to IPASIR:

- `ipamir_add_soft_lit` declares a soft literal b with weight w
  - if literal b already declared soft, changes its weight
- `ipamir_val_obj` computes the cost of the current solution

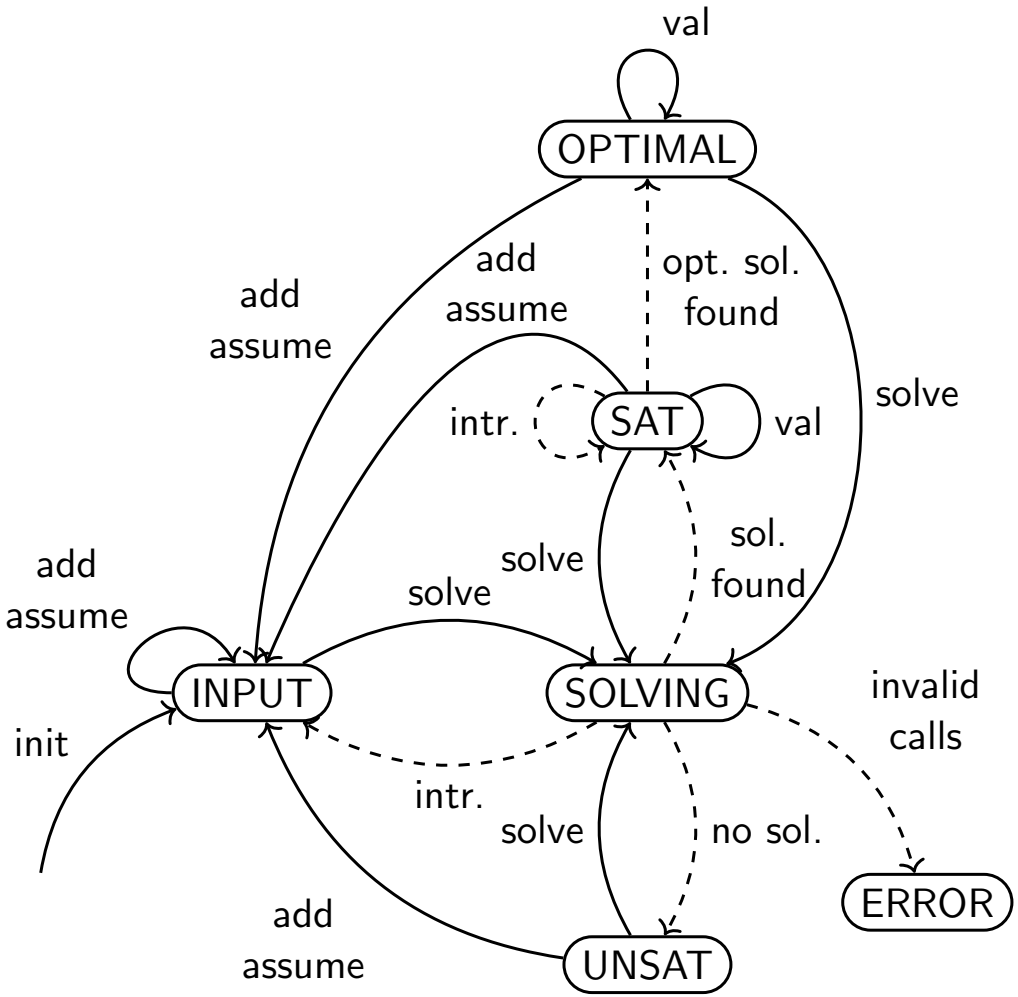
```
// Construct a MaxSAT solver and return a pointer to it.
void * ipamir_init ();
// Deallocate all resources of the MaxSAT solver.
void ipamir_release (void * solver);
// Add a literal to a hard clause or finalize the clause with zero.
void ipamir_add_hard (void * solver, int32_t lit_or_zero);
// 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));
```

Functions declared in the IPAMIR header.

In contrast to IPASIR:

- `ipamir_add_soft_lit` declares a soft literal b with weight w
  - if literal b already declared soft, changes its weight
- `ipamir_val_obj` computes the cost of the current solution

IPAMIR: an incremental interface for MaxSAT



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.

- A *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

Davies and Bacchus [2011, 2013]

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

- A *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

Davies and Bacchus [2011, 2013]

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

- A *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

Davies and Bacchus [2011, 2013]

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

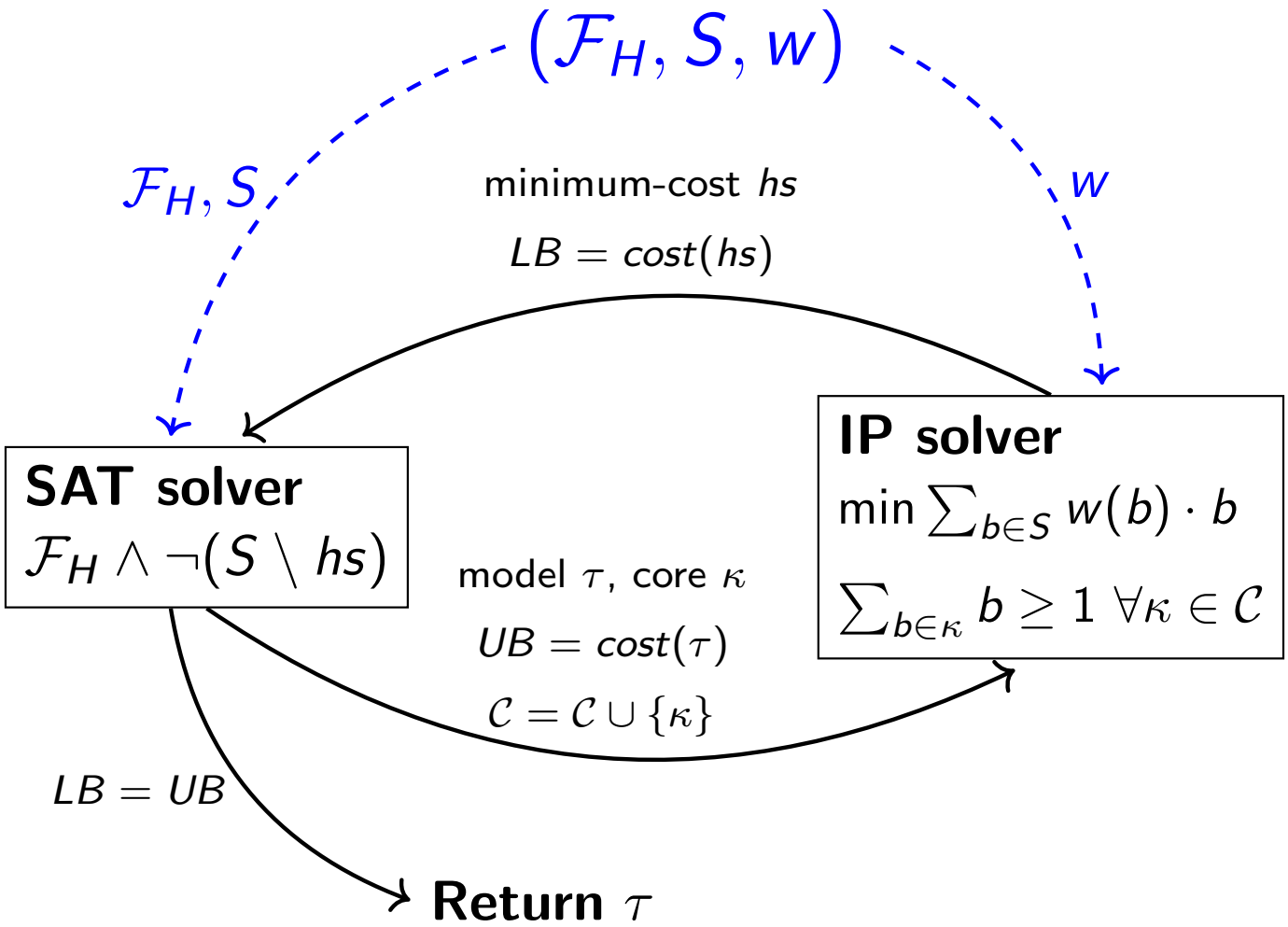
- A *core* is a clause over soft literals entailed by the hard clauses.
  - SAT solver as *core extractor*
- *hs* is a *hitting set* over a set of cores $\mathcal{C}$ if *hs* intersects each $\kappa \in \mathcal{C}$
  - *cost* of a hitting set determined by weights of soft literals
  - IP solver for 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

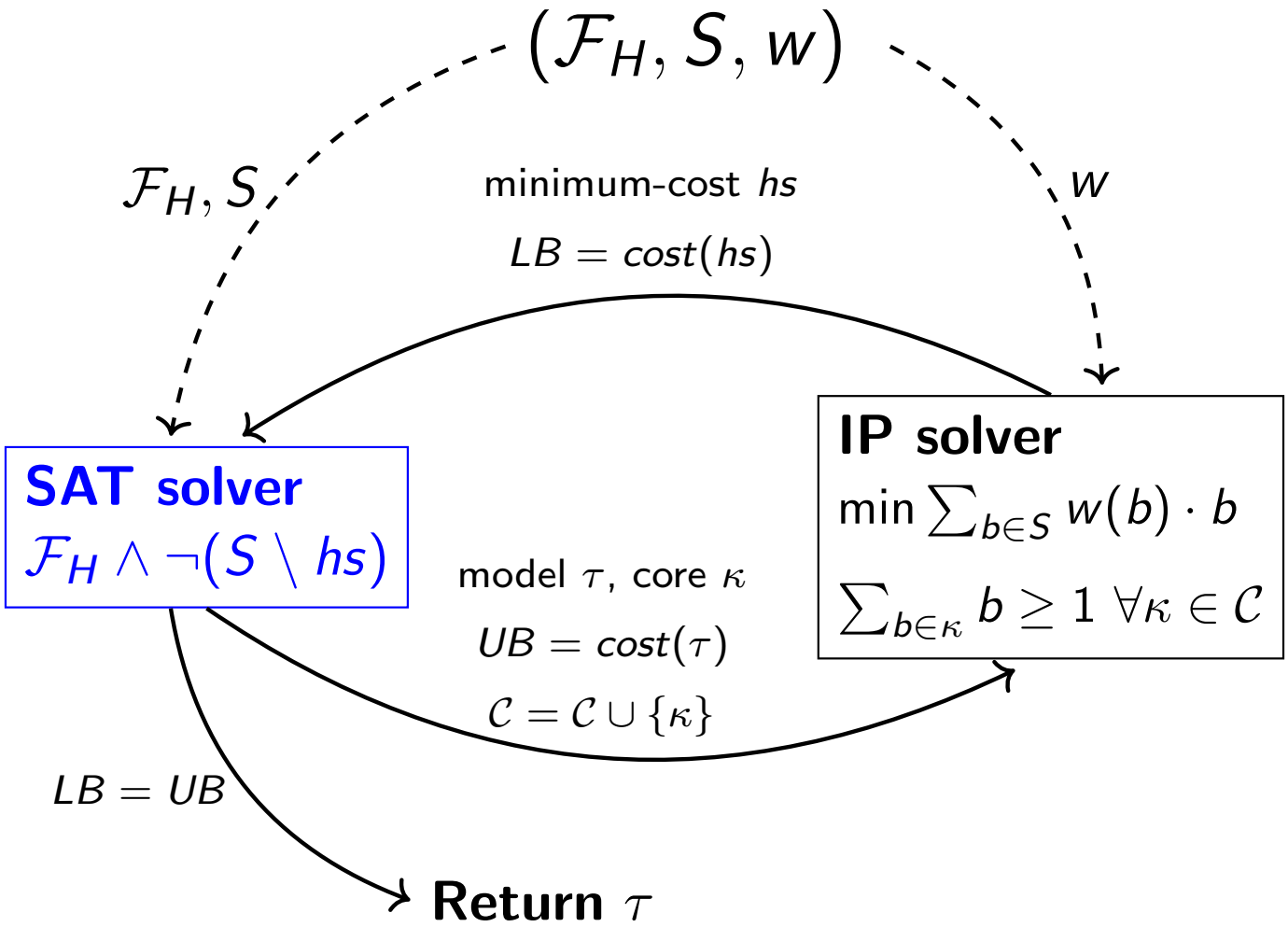
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



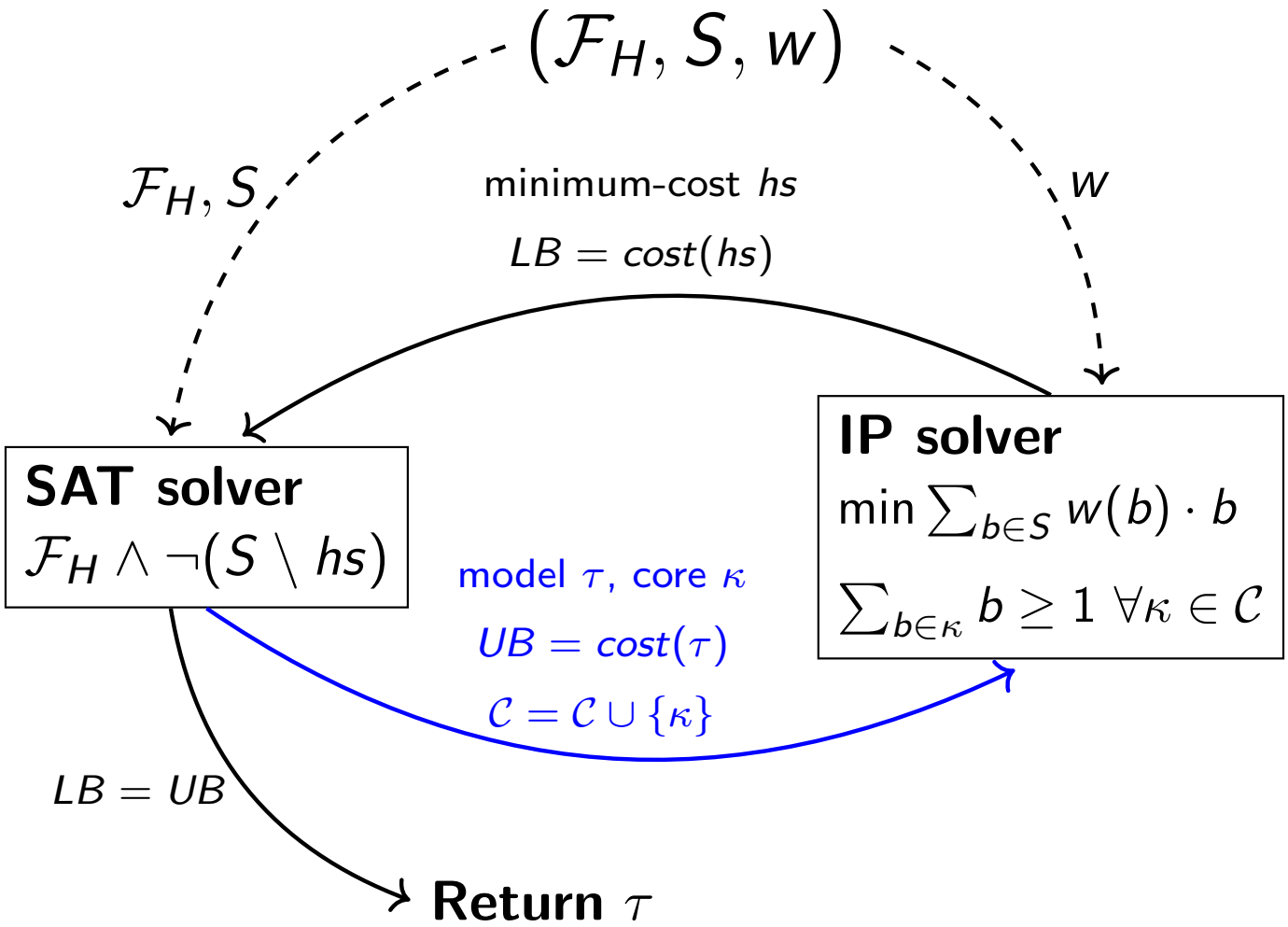
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



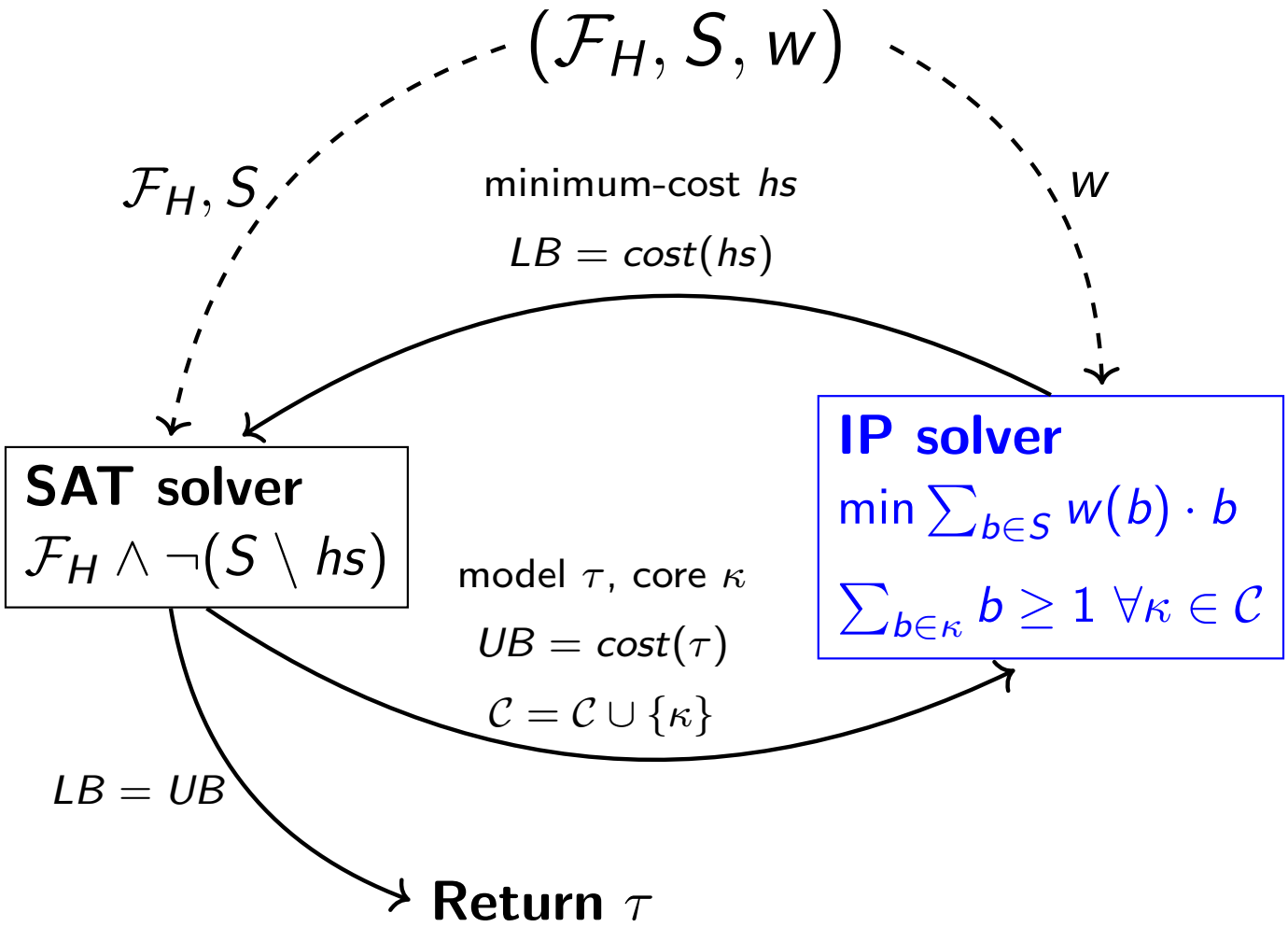
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



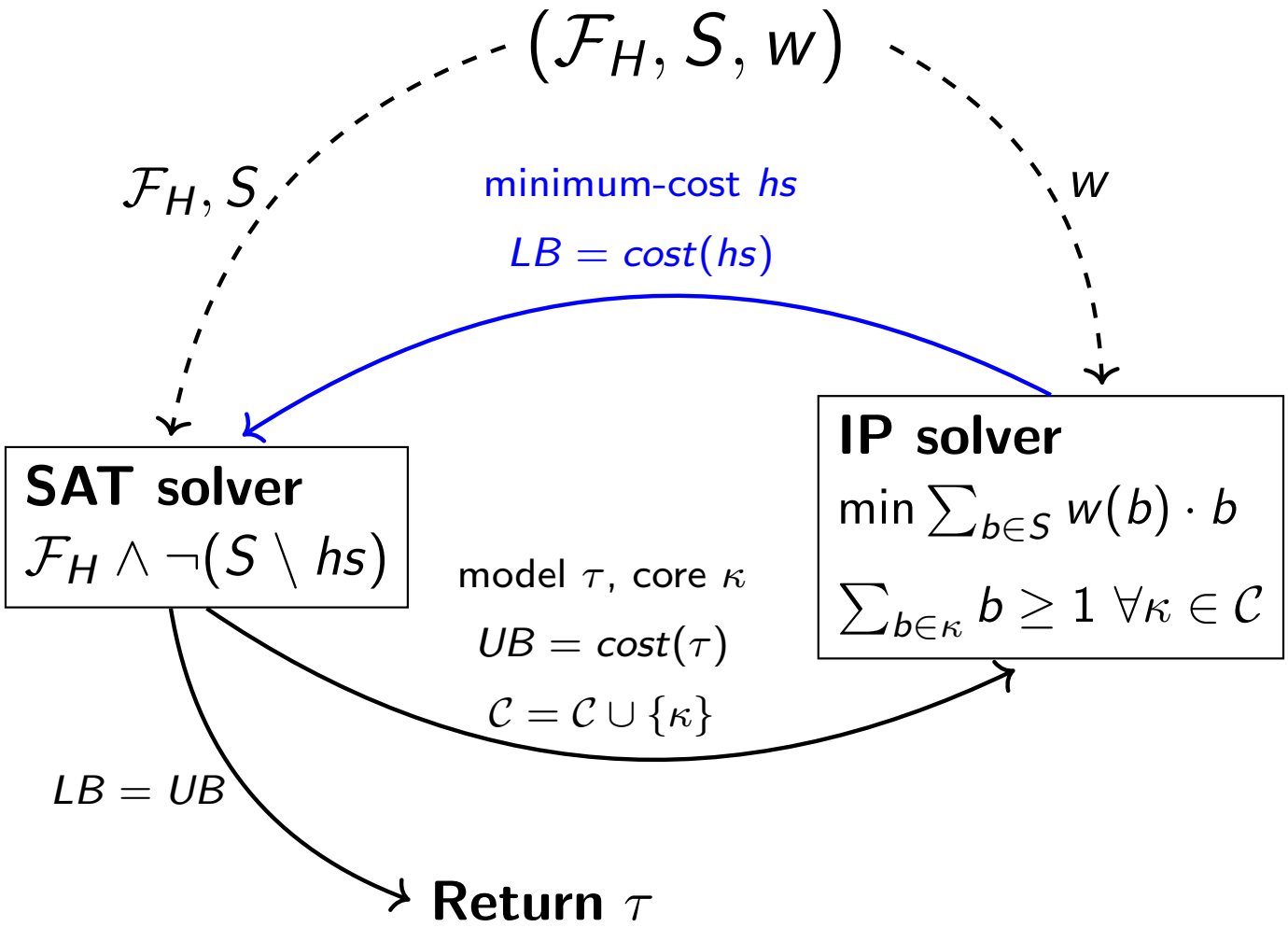
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



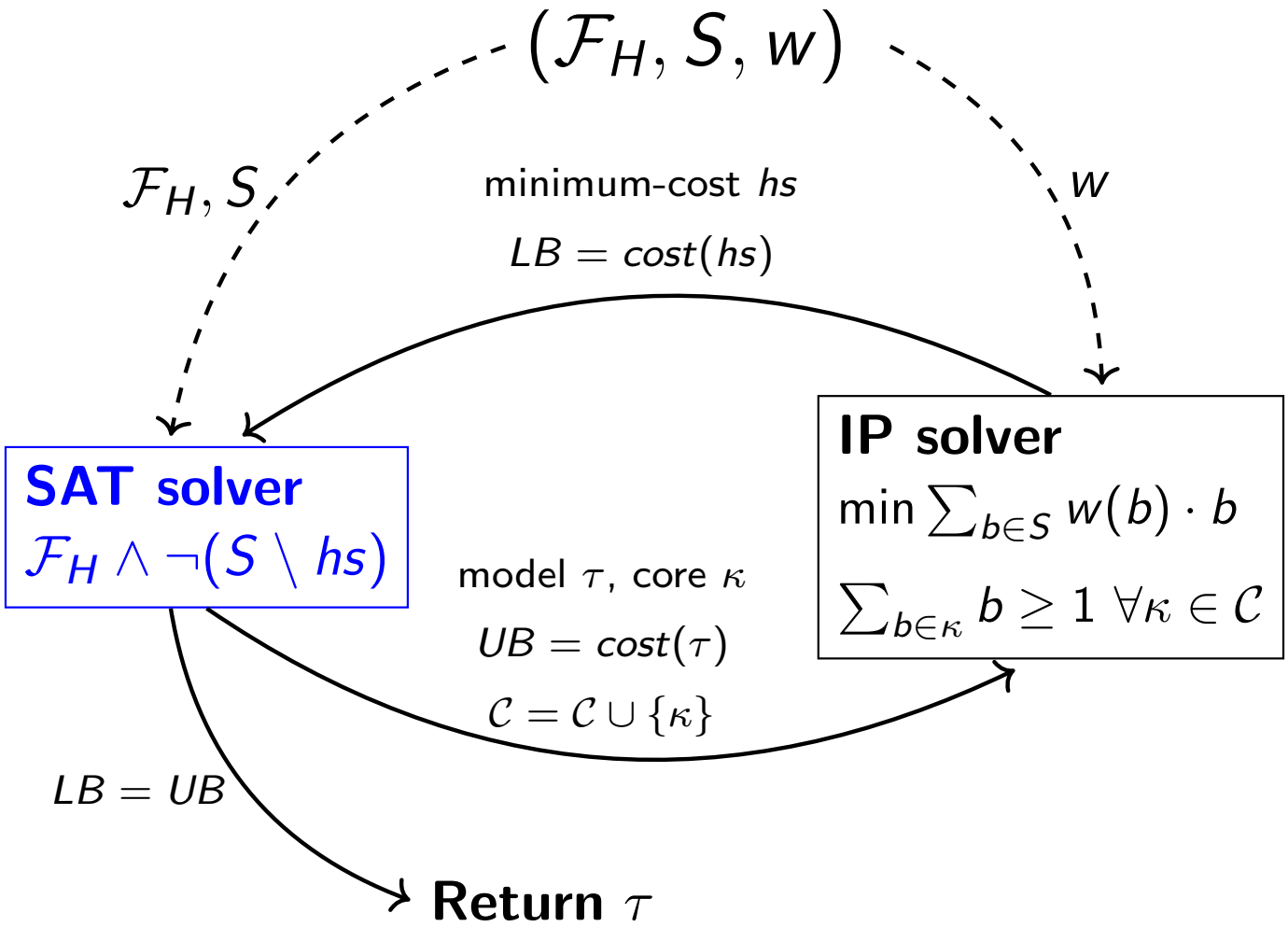
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



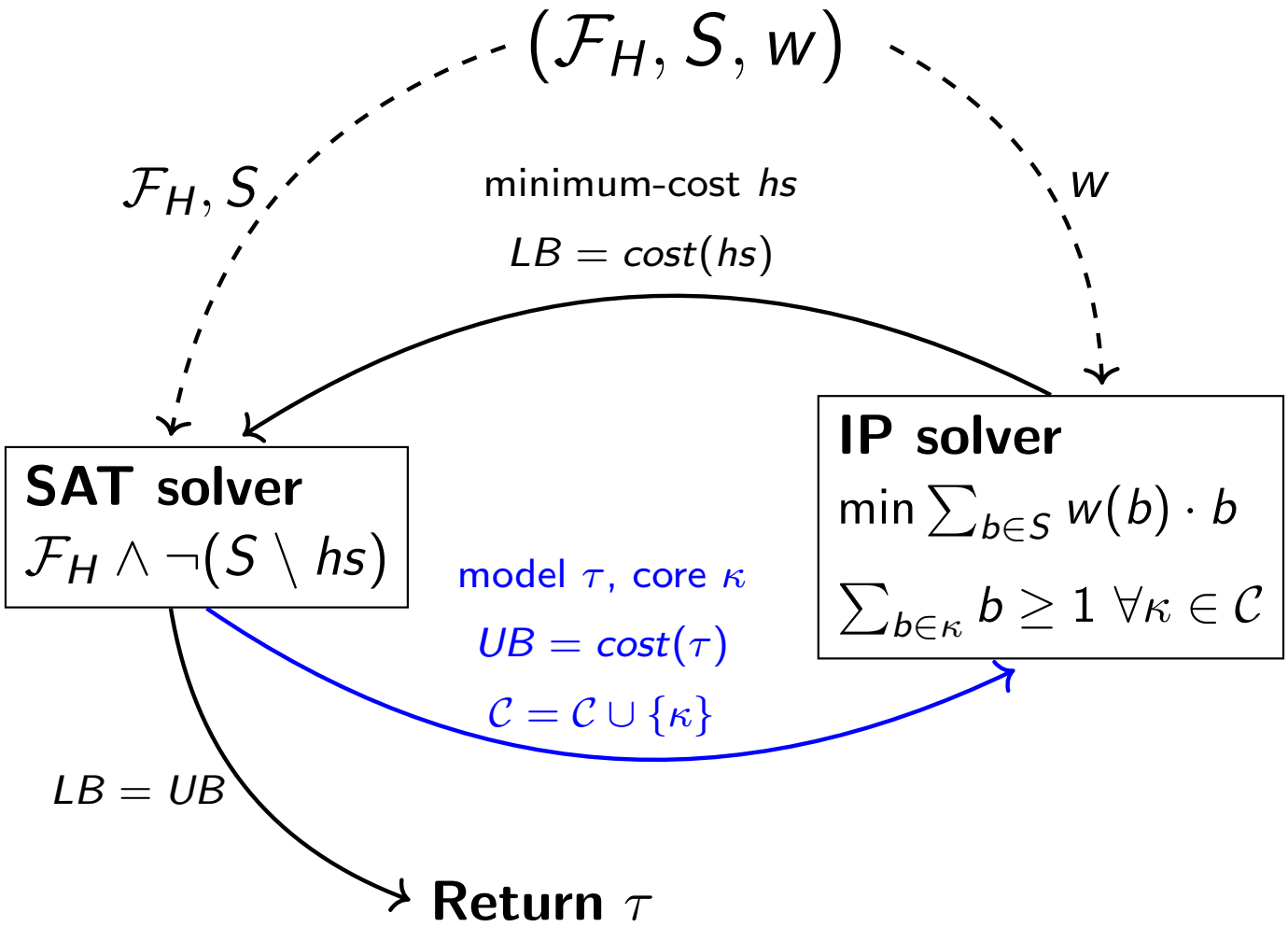
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



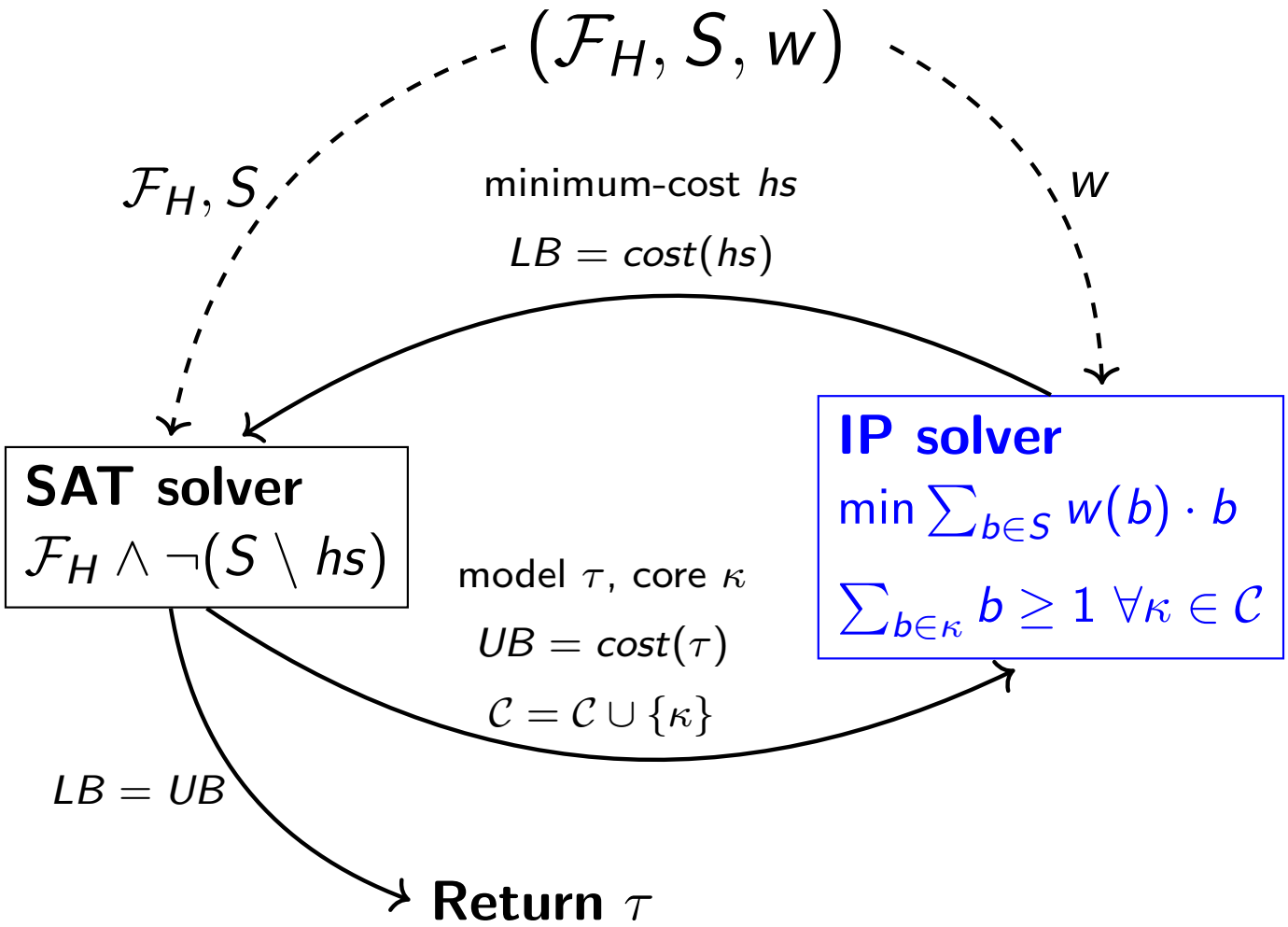
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



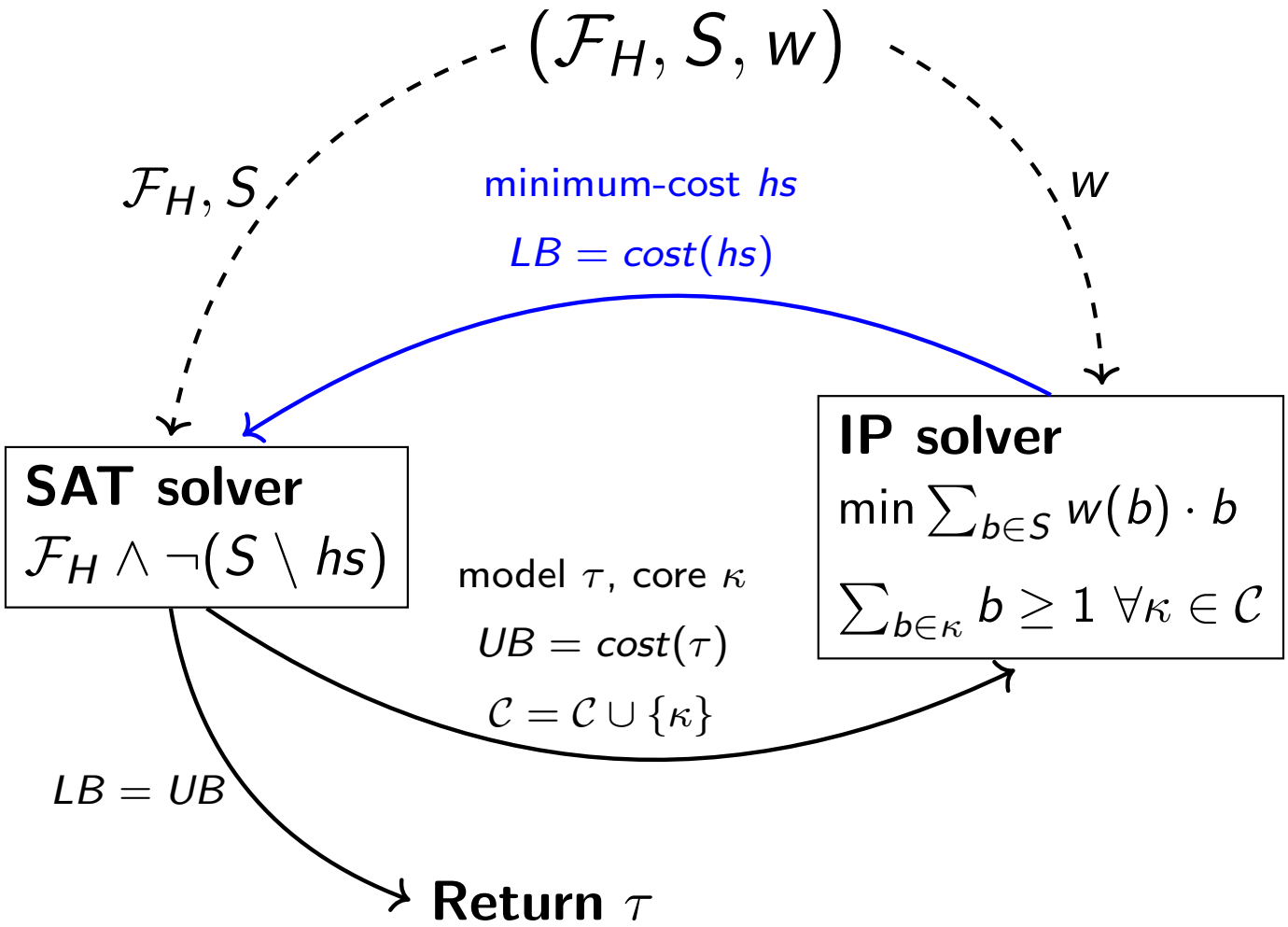
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



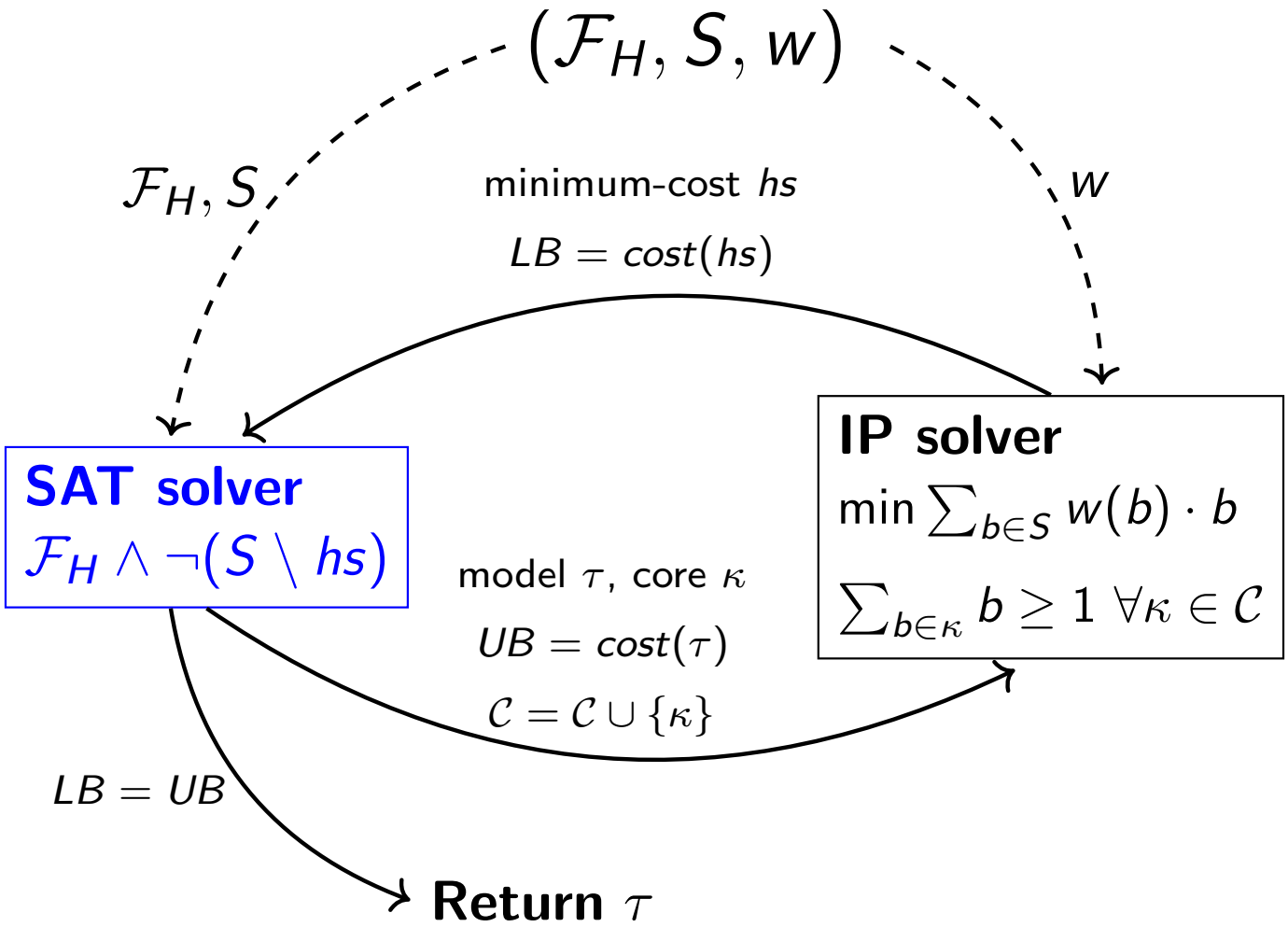
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



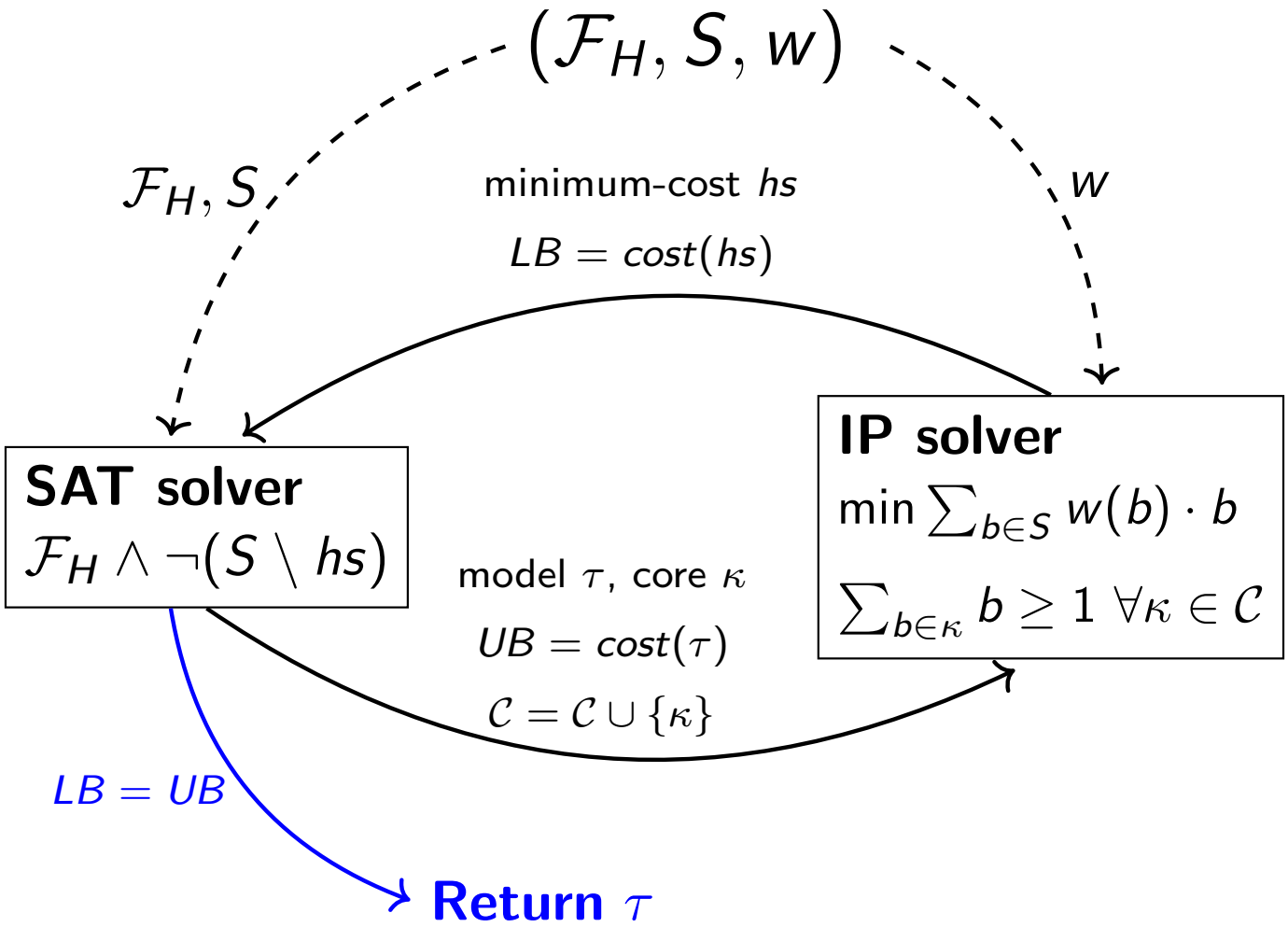
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



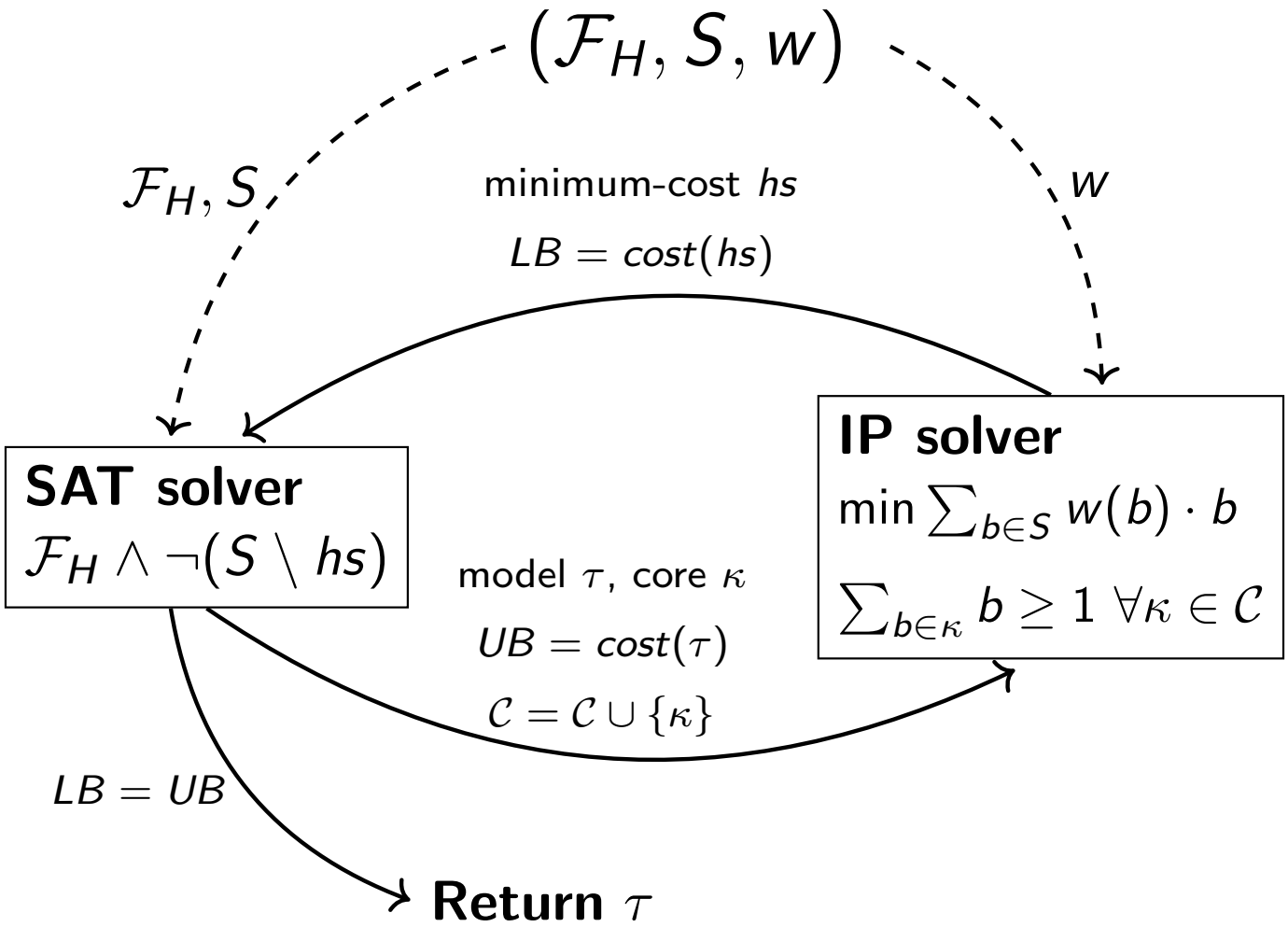
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



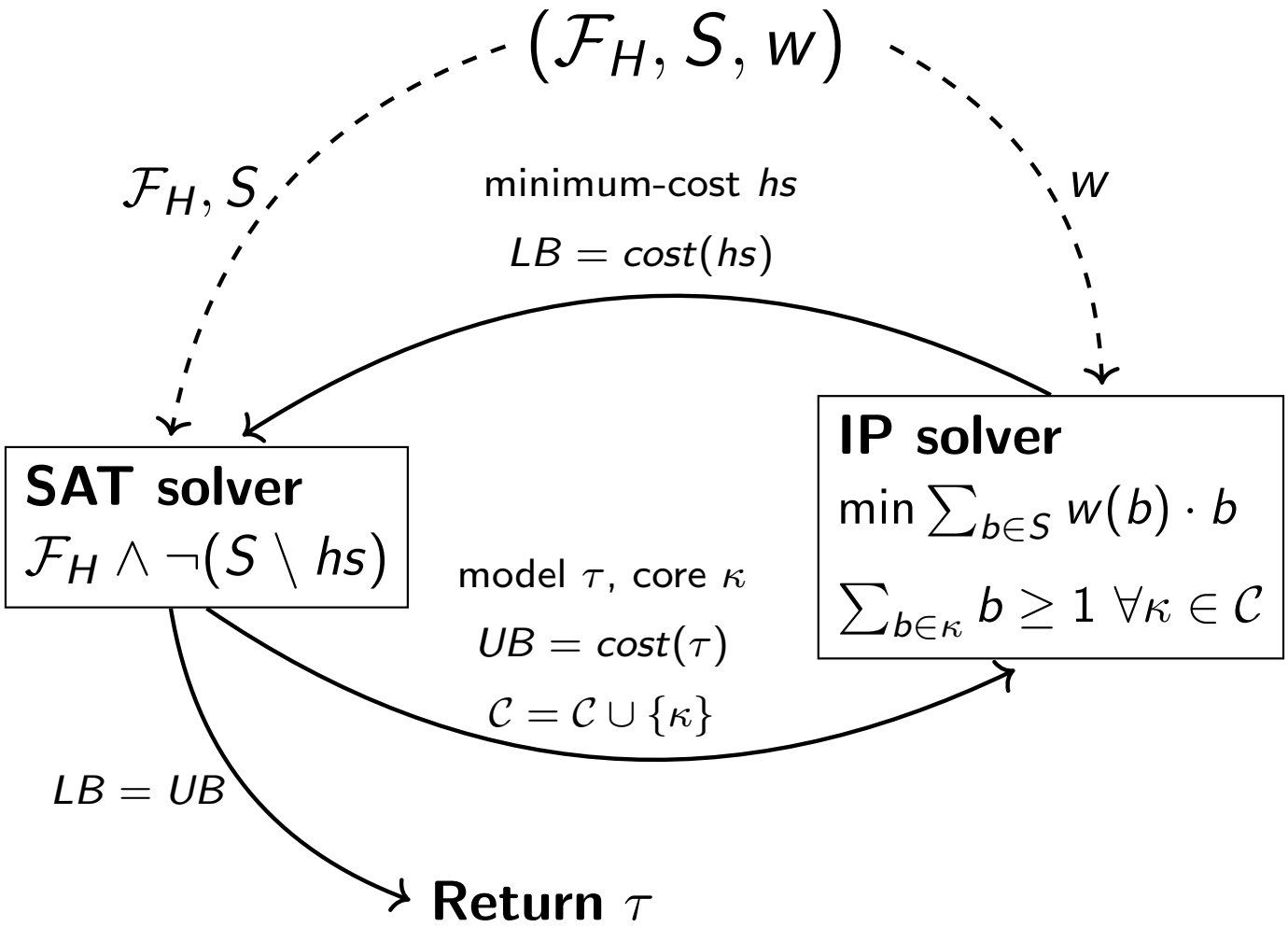
Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



Implicit Hitting Set (IHS) based MaxSAT solving

Davies and Bacchus [2011, 2013]



Note: $\neg(S \setminus hs)$ passed as assumptions to the SAT solver.

Incremental IHS

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?

Incremental IHS

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?

Incremental IHS for solving under assumptions

In theory

Main idea: pass user-provided assumptions A along with IHS solving assumptions $\neg(S \setminus hs)$ to the internal SAT solver

- if $a \in A \cap S$, do not include $\neg a$ as assumption from $\neg(S \setminus hs)$
- 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 $\mathcal{F}_H$.
The *restriction* of a conditional core is $\kappa^a \setminus \neg A$.

Incremental IHS for solving under assumptions

In theory

Main idea: pass user-provided assumptions A along with IHS solving assumptions $\neg(S \setminus hs)$ to the internal SAT solver

- if $a \in A \cap S$, do not include $\neg a$ as assumption from $\neg(S \setminus hs)$
- 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 $\mathcal{F}_H$.
The *restriction* of a conditional core is $\kappa^a \setminus \neg A$.

Incremental IHS for solving under assumptions

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.

Incremental IHS for solving under assumptions

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.

Incremental IHS for solving under assumptions

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.

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.

- **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
  - fixed literals need to be handled correctly
  - no pure literal elimination can be performed
- **Other techniques** must be modified to preserve correctness
  - reduced cost fixing Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores Berg, Bacchus, and Poole [2020]

End result: **iMaxHS** (incremental MaxHS)

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.

- **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
  - fixed literals need to be handled correctly
  - no pure literal elimination can be performed
- **Other techniques** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

End result: **iMaxHS** (incremental MaxHS)

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.

- **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
  - fixed literals need to be handled correctly
  - no pure literal elimination can be performed
- **Other techniques** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

End result: **iMaxHS** (incremental MaxHS)

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.

- **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
  - fixed literals need to be handled correctly
  - no pure literal elimination can be performed
- **Other techniques** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

End result: **iMaxHS** (incremental MaxHS)

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.

- **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
  - fixed literals need to be handled correctly
  - no pure literal elimination can be performed
- **Other techniques** must be modified to preserve correctness
  - reduced cost fixing — Bacchus, Hyttinen, Järvisalo, and Saikko [2017]
  - abstract cores — Berg, Bacchus, and Poole [2020]

End result: **iMaxHS** (incremental MaxHS)

Empirical evaluation

Optimizing under assumptions

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

Optimizing under assumptions

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

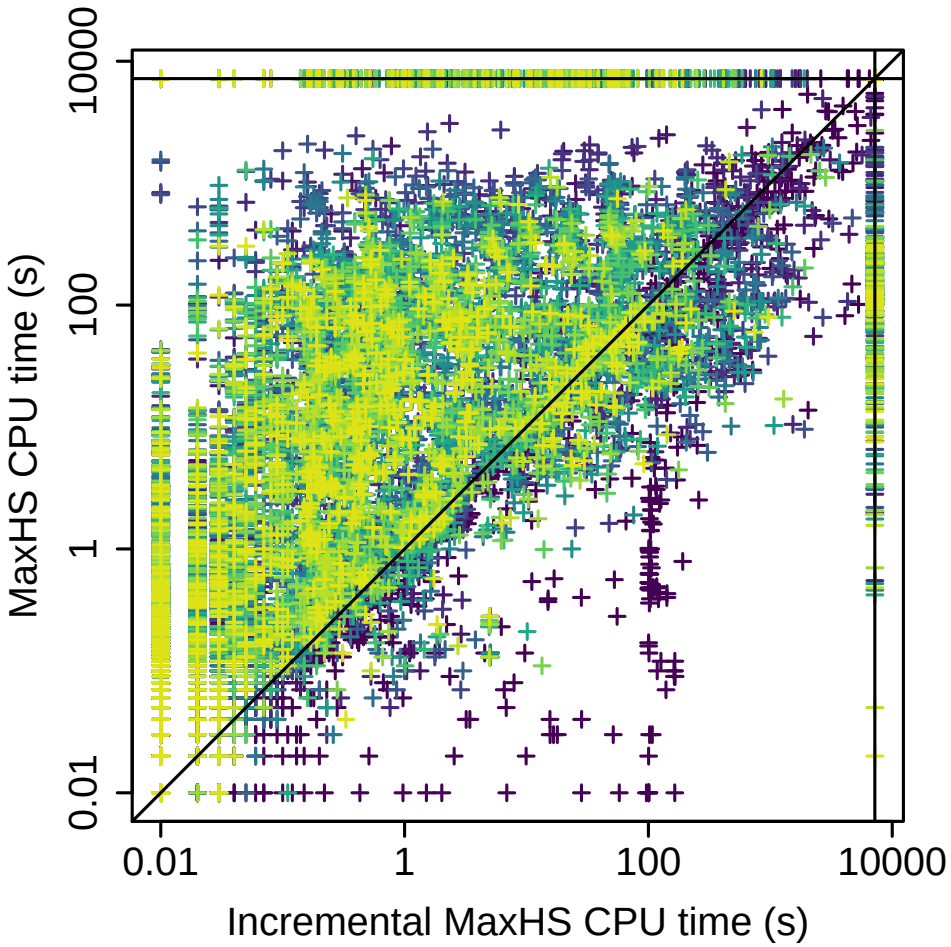
Benchmark setup

- 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

Optimizing under assumptions

Increased performance gained by preserving cores:



Summary

Contributions

- **IPAMIR:** incremental API for MaxSAT
  - specifies various forms of incrementality in MaxSAT
  - provides a standard interface to facilitate the development of solvers and applications
- **iMaxHS:** fully-fledged incremental MaxSAT solver
  - supports all IPAMIR functionality
  - internal SAT solver used without reinitializing
  - cores preserved between solver 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.

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.

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.

Bibliography II

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.