Computing with SAT Oracles

Joao Marques-Silva, Alexey Ignatiev & Antonio Morgado

University of Lisbon, Portugal

August 12, 2019 | IJCAI Tutorial T20

Computing with SAT Oracles

Joao Marques-Silva, Alexey Ignatiev & Antonio Morgado

University of Lisbon, Portugal

August 12, 2019 | IJCAI Tutorial T20

What is SAT?

- SAT is the decision problem for propositional logic
  - Well-formed propositional formulas, with variables, logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and parenthesis: $($, $)$
  - Often restricted to Conjunctive Normal Form (CNF)

What is SAT?

- SAT is the decision problem for propositional logic
  - Well-formed propositional formulas, with variables, logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and parenthesis: $($, $)$
  - Often restricted to Conjunctive Normal Form (CNF)
  - Goal:
    Decide whether formula has a satisfying assignment

What is SAT?

- SAT is the decision problem for propositional logic
  - Well-formed propositional formulas, with variables, logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and parenthesis: $($, $)$
  - Often restricted to Conjunctive Normal Form (CNF)
  - Goal:
    Decide whether formula has a satisfying assignment

- SAT is NP-complete [Coo71]

The CDCL SAT disruption

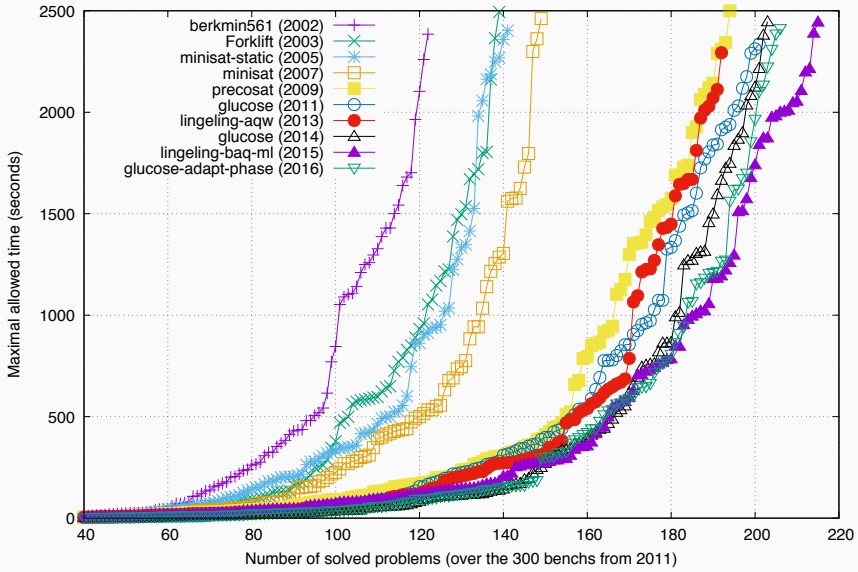
- CDCL SAT solving is a success story of Computer Science

The CDCL SAT disruption

- CDCL SAT solving is a success story of Computer Science
  - Conflict-Driven Clause Learning (CDCL)
  - (CDCL) SAT has impacted many different fields
  - Hundreds (thousands?) of practical applications

CDCL SAT solver (continued) improvement

[Source: Simon 2015]



How good are CDCL SAT solvers?

Demos

How good are CDCL SAT solvers?

Demos

- Sample SAT of solvers:
  1. POSIT: state of the art DPLL SAT solver in 1995
  2. GRASP: first CDCL SAT solver, state of the art 1995∼2000
  3. Minisat: CDCL SAT solver, state of the art until the late 00s
  4. Glucose: modern state of the art CDCL SAT solver
  5. ...

How good are CDCL SAT solvers?

Demos

- Sample SAT of solvers:
  1. POSIT: state of the art DPLL SAT solver in 1995
  2. GRASP: first CDCL SAT solver, state of the art 1995∼2000
  3. Minisat: CDCL SAT solver, state of the art until the late 00s
  4. Glucose: modern state of the art CDCL SAT solver
  5. ...
- Example 1: model checking example (from IBM)
- Example 2: cooperative path finding (CPF)

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem

- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents
  - Formula w/ 2946190 variables!

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

How good are SAT solvers? – an example

- Cooperative pathfinding (CPF)
  - N agents on some grid/graph
  - Start positions
  - Goal positions
  - Minimize makespan
  - Restricted planning problem
- Concrete example
  - Gaming grid
  - 1039 vertices
  - 1928 edges
  - 100 agents
  - Formula w/ 2946190 variables!
- **Note:** In the early 90s, SAT solvers could solve formulas with a few hundred variables!

```
*** tracker: a pathfinding tool ***

Initialization ... CPU Time: 0.004711
Number of variables: 113315
Tentative makespan 1
Number of variables: 226630
Number of assumptions: 1
c Running SAT solver ... CPU Time: 0.718112
c Done running SAT solver ... CPU Time: 0.830099
No solution for makespan 1
Elapsed CPU Time: 0.830112
Tentative makespan 2
Number of variables: 339945
Number of assumptions: 1
c Running SAT solver ... CPU Time: 1.27113
c Done running SAT solver ... CPU Time: 1.27114
No solution for makespan 2
Elapsed CPU Time: 1.27114
...
...
Tentative makespan 24
Number of variables: 2832875
Number of assumptions: 1
c Running SAT solver ... CPU Time: 11.8653
c Done running SAT solver ... CPU Time: 11.8653
No solution for makespan 24
Elapsed CPU Time: 11.8653
Tentative makespan 25
Number of variables: 2946190
Number of assumptions: 1
c Running SAT solver ... CPU Time: 12.3491
c Done running SAT solver ... CPU Time: 16.6882
Solution found for makespan 25
Elapsed CPU Time: 16.6995
```

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):

Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ **!**
  - **Obs:** SAT solvers in the late 90s (but formula dependent)

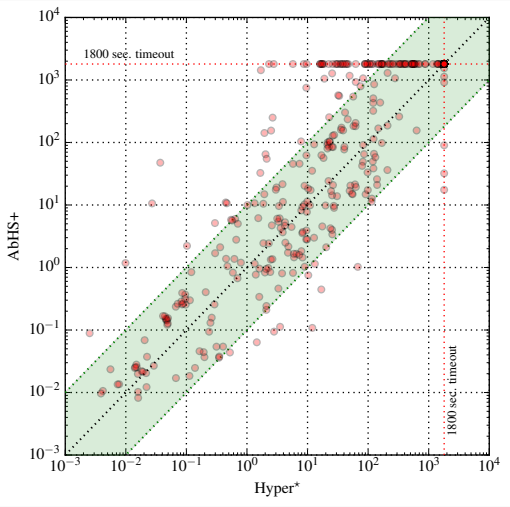
Grasping the search space ...

- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ **!**
  - **Obs:** SAT solvers in the late 90s (but formula dependent)
- Search space with 2832875 propositional variables (worst case):

Grasping the search space ...

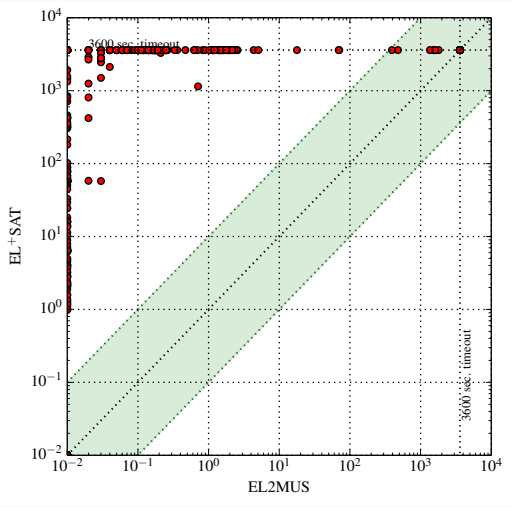
- Number of seconds since the Big Bang: $\approx 10^{17}$
- Number of fundamental particles in observable universe: $\approx 10^{80}$ (or $\approx 10^{85}$)
- Search space with 15775 propositional variables (worst case):
  - # of assignments to 15775 variables: $> 10^{4748}$ **!**
  - **Obs:** SAT solvers in the late 90s (but formula dependent)
- Search space with 2832875 propositional variables (worst case):
  - # of assignments to $> 2.8 \times 10^6$ variables: $\gg 10^{840000}$ **!!**
  - **Obs:** SAT solvers at present (but formula dependent)

SAT can make the difference – propositional abduction



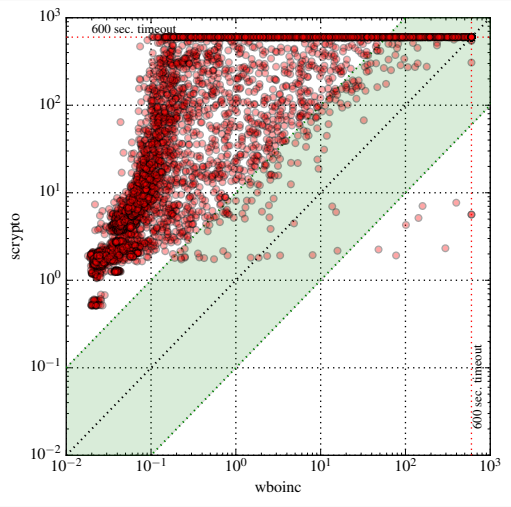
- Propositional abduction instances [IMM16]
  - Implicit hitting set dualization (IHSD)

SAT can make the difference – axiom pinpointing



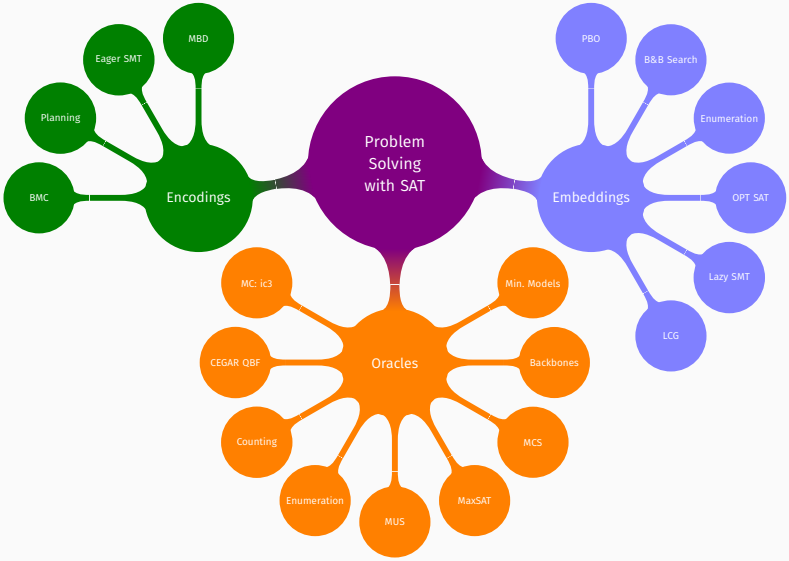
- $\mathcal{EL}^+$ medical ontologies [AMM15]
  - Minimal unsatisfiability (MUSes) & maximal satisfiability (MCSes) & Enumeration

SAT can make the difference – model based diagnosis

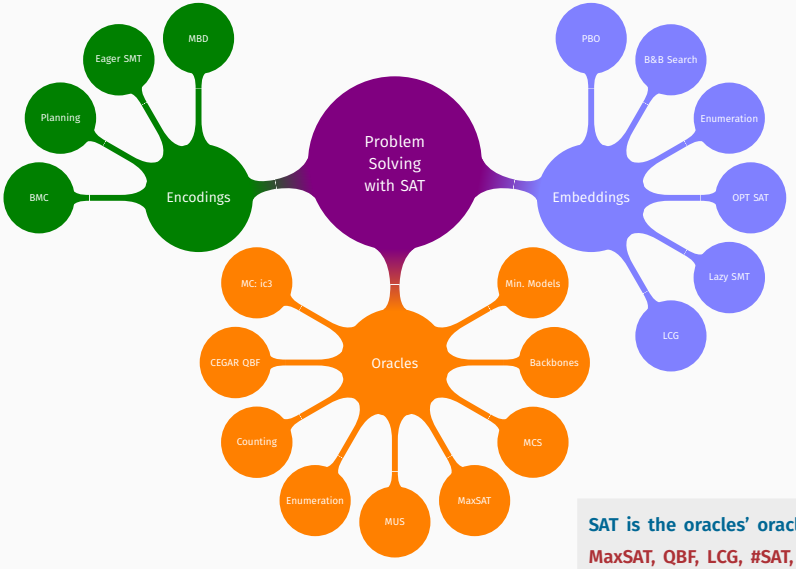


- Model-based diagnosis problem instances [MJIM15]
  - Maximum satisfiability (MaxSAT)

CDCL SAT is ubiquitous in problem solving



CDCL SAT is ubiquitous in problem solving



SAT is the oracles' oracle:
MaxSAT, QBF, LCG, #SAT, SMT, ASP, FOL, ...

Tutorial organization

- Part #1: Overview & basic definitions – Joao
- Part #2: Modern CDCL SAT solvers – Alexey
- Part #3: Modeling with propositional logic – Antonio

Tutorial organization

- Part #1: Overview & basic definitions – Joao
- Part #2: Modern CDCL SAT solvers – Alexey
- Part #3: Modeling with propositional logic – Antonio
- Part #4: Problem solving with SAT oracles – Joao
- Part #5: Sample of applications – Alexey
- Part #6: Applications in proof complexity – Antonio

Tutorial organization

- Part #1: Overview & basic definitions – Joao
- Part #2: Modern CDCL SAT solvers – Alexey
- Part #3: Modeling with propositional logic – Antonio
- Part #4: Problem solving with SAT oracles – Joao
- Part #5: Sample of applications – Alexey
- Part #6: Applications in proof complexity – Antonio
- Part #7: A glimpse of the future – Joao

1 Basic Definitions

Preliminaries

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT

Preliminaries

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT
- Example:

$$\mathcal{F} \triangleq (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee b) \wedge (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

  - Example models:

Preliminaries

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT
- Example:

$$\mathcal{F} \triangleq (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee b) \wedge (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

  - Example models:
    - $\{r, s, a, b, c, d\}$

Preliminaries

- Variables: $w, x, y, z, a, b, c, \ldots$
- Literals: $w, \bar{x}, \bar{y}, a, \ldots$, but also $\neg w, \neg y, \ldots$
- Clauses: disjunction of literals or set of literals
- Formula: conjunction of clauses or set of clauses
- Model (satisfying assignment): partial/total mapping from variables to $\{0, 1\}$ that satisfies formula
- Each clause can be satisfied, falsified, but also unit, unresolved
- Formula can be SAT/UNSAT
- Example:

$$\mathcal{F} \triangleq (r) \wedge (\bar{r} \vee s) \wedge (\bar{w} \vee a) \wedge (\bar{x} \vee b) \wedge (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

  - Example models:
    - $\{r, s, a, b, c, d\}$
    - $\{r, s, \bar{x}, y, \bar{w}, z, \bar{a}, b, c, d\}$

Resolution

- Resolution rule: [DP60, Rob65]

$$\frac{(\alpha \vee x) \qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

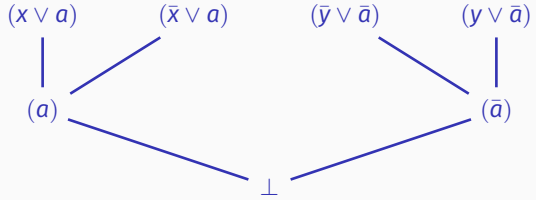
  - Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60, Rob65]

$$\frac{(\alpha \vee x) \qquad (\beta \vee \bar{x})}{(\alpha \vee \beta)}$$

  - Complete proof system for propositional logic



  - Extensively used with (CDCL) SAT solvers

Unit propagation

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

Unit propagation

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

Unit propagation

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

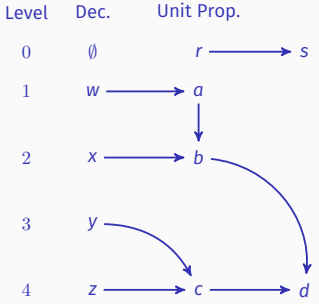
- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

- **Unit clause rule**: if clause is unit, its sole literal **must** be satisfied

Unit propagation

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

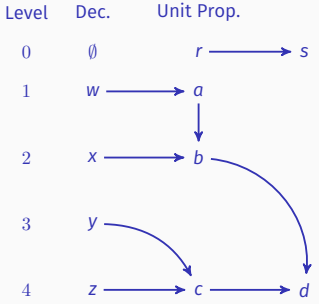
- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$



- **Unit clause rule**: if clause is unit, its sole literal **must** be satisfied

Unit propagation

$$\begin{aligned}\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$



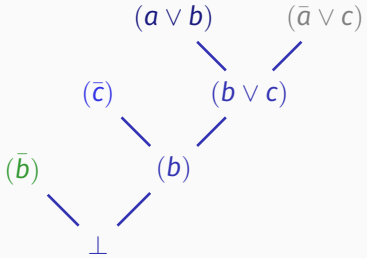
- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

- **Unit clause rule**: if clause is unit, its sole literal **must** be satisfied

- Additional definitions:
  - Antecedent (or reason) of an implied assignment
    - $(\bar{b} \vee \bar{c} \vee d)$ for d
  - Associate assignment with decision levels
    - $w = 1 \,@\, 1, x = 1 \,@\, 2, y = 1 \,@\, 3, z = 1 \,@\, 4$
    - $r = 1 \,@\, 0, d = 1 \,@\, 4$, ...

Resolution proofs

- Refutation of unsatisfiable formula by iterated resolution operations yields resolution proof
- An example:
  $\mathcal{F} = (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$
- Resolution proof:



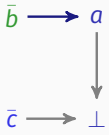
- Modern SAT solvers can generate resolution proofs from learned clauses [ZM03]

Unsatisfiable cores & proof traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	$\bar{b} \rightarrow a$
		$\bar{c} \rightarrow \bot$



Implication graph with conflict

Unsatisfiable cores & proof traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$

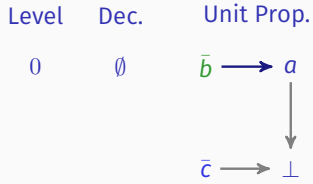
Level	Dec.	Unit Prop.
0	$\emptyset$	$\bar{b} \longrightarrow a$
		$\bar{c} \longrightarrow \bot$

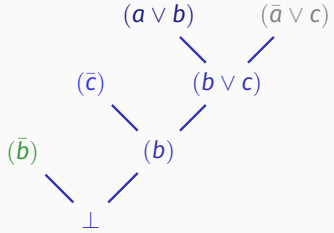
Proof trace $\bot$: $(\bar{a} \vee c)$ $(a \vee b)$ $(\bar{c})$ $(\bar{b})$

Unsatisfiable cores & proof traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$





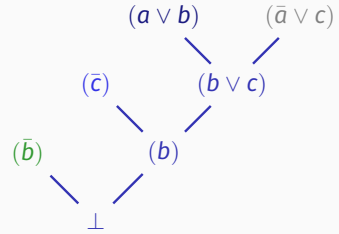
Resolution proof follows structure of conflicts

Unsatisfiable cores & proof traces

- CNF formula:

$$\mathcal{F} \quad = \quad (\bar{c}) \wedge (\bar{b}) \wedge (\bar{a} \vee c) \wedge (a \vee b) \wedge (a \vee \bar{d}) \wedge (\bar{a} \vee \bar{d})$$

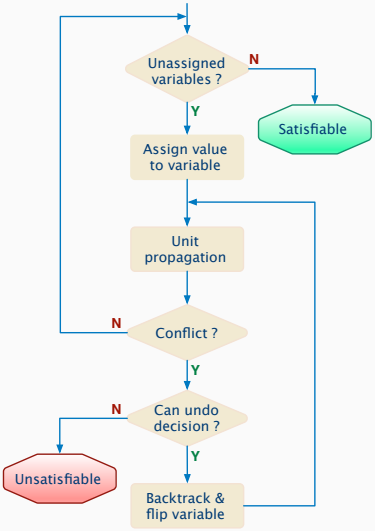
Level	Dec.	Unit Prop.
0	$\emptyset$	$\bar{b} \rightarrow a$, $\bar{c} \rightarrow \bot$



Unsatisfiable subformula (core): $(\bar{c}), (\bar{b}), (\bar{a} \vee c), (a \vee b)$

The DPLL algorithm

[DP60, DLL62]

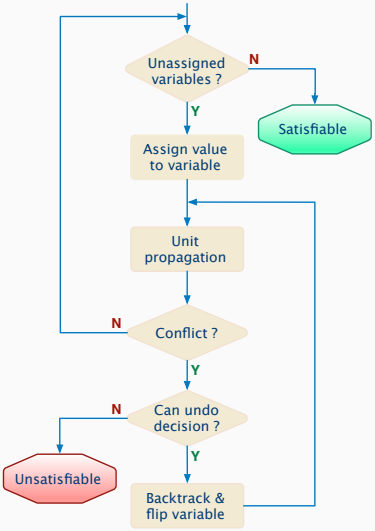


- Optional: pure literal rule

The DPLL algorithm

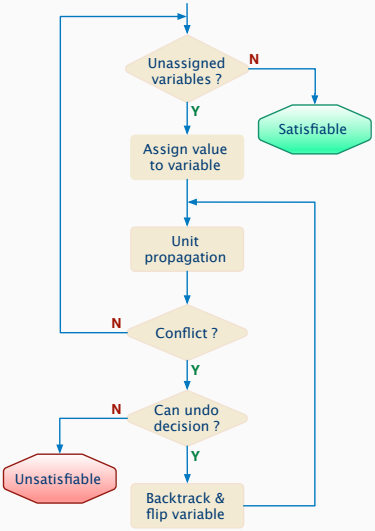
[DP60, DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



- Optional: pure literal rule

The DPLL algorithm

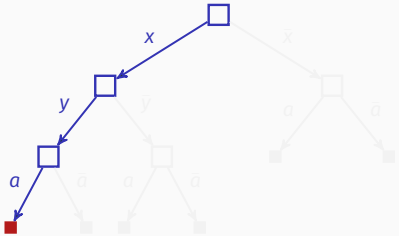


- Optional: pure literal rule

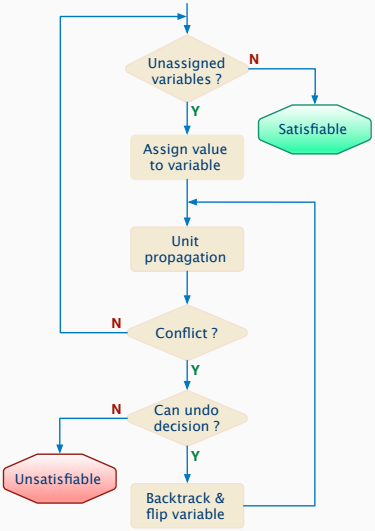
[DP60, DLL62]

$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	a	$\rightarrow b \rightarrow \bot$



The DPLL algorithm

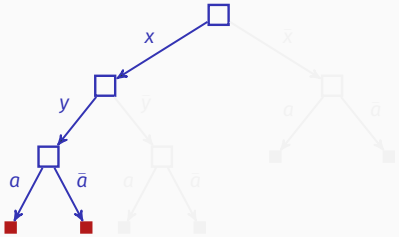


- Optional: pure literal rule

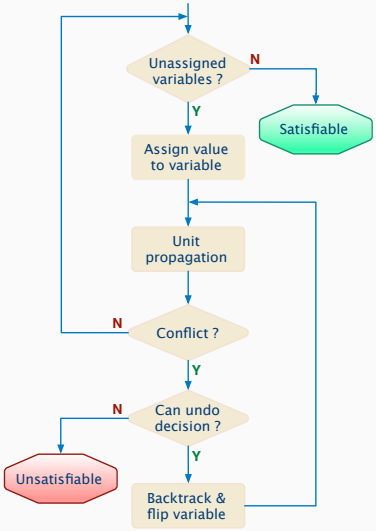
[DP60, DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\bar{a}$	$\bar{b}$ $\quad \bot$



The DPLL algorithm

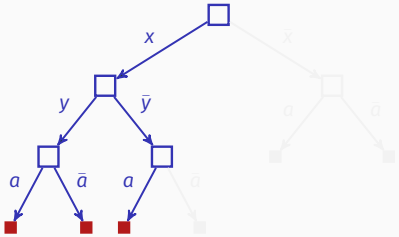


- Optional: pure literal rule

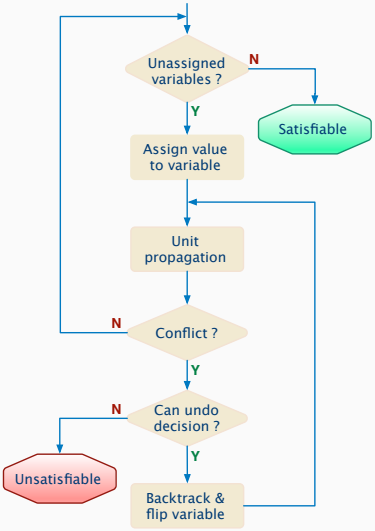
[DP60, DLL62]

$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	a	$\rightarrow b \rightarrow \bot$



The DPLL algorithm

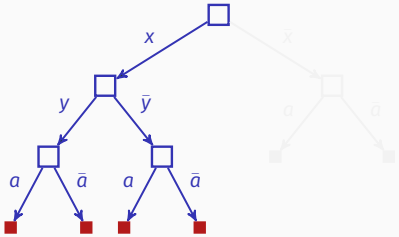


- Optional: pure literal rule

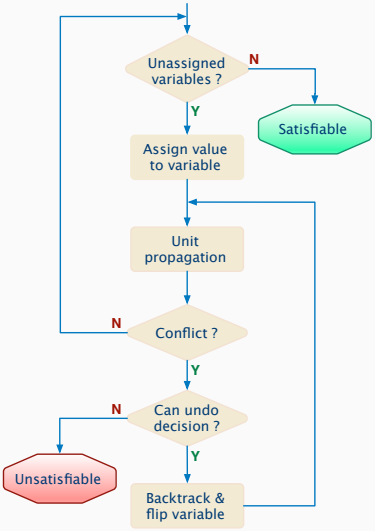
[DP60, DLL62]

$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	$\bar{a}$	$\bar{b} \rightarrow \bot$



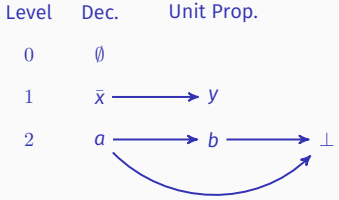
The DPLL algorithm

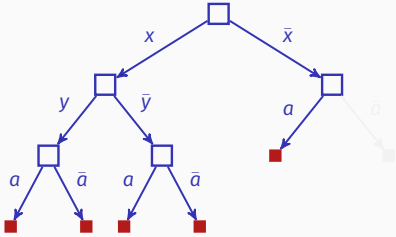


- Optional: pure literal rule

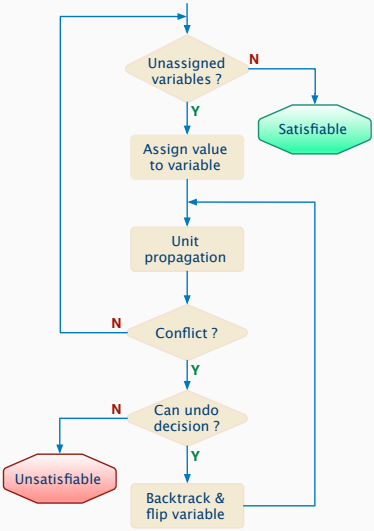
[DP60, DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$





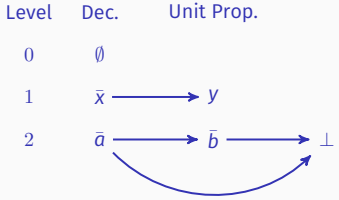
The DPLL algorithm

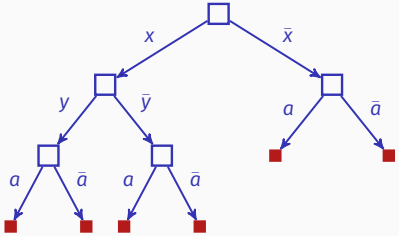


- Optional: pure literal rule

[DP60, DLL62]

$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$





2 CDCL SAT Solvers

What is a CDCL SAT solver?

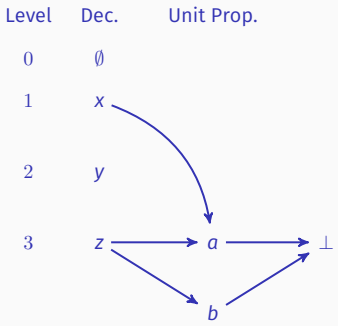
- Extend DPLL SAT solver with: [DP60, DLL62]
  - Clause learning & non-chronological backtracking [MS95, MSS96b, MSS99]
  - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
  - Lazy data structures
  - Conflict-guided branching
  - ...

What is a CDCL SAT solver?

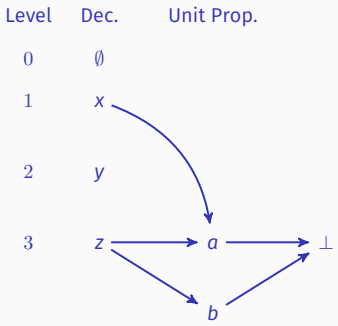
- Extend DPLL SAT solver with: [DP60, DLL62]
  - Clause learning & non-chronological backtracking [MS95, MSS96b, MSS99]
    - Exploit UIPs [MS95, MSS99, ZMMM01, SSS12]
    - Minimize learned clauses [SB09, Gel09, LLX+17]
    - Opportunistically delete clauses [MSS96b, MSS99, GN02, AS09]
  - Search restarts [GSC97, BMS00, Hua07, Bie08, LOM+18]
  - Lazy data structures
    - Watched literals [MMZ+01]
  - Conflict-guided branching
    - Lightweight branching heuristics [MMZ+01]
    - Phase saving [PD07]
  - ...

Clause Learning, UIPs & Minimization

Clause learning



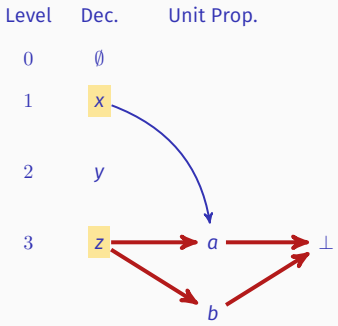
Clause learning



- Analyze conflict

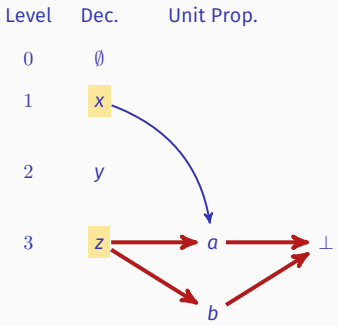
[MS95, MSS96a, MSS96a, MSS96b, MSS99]

Clause learning



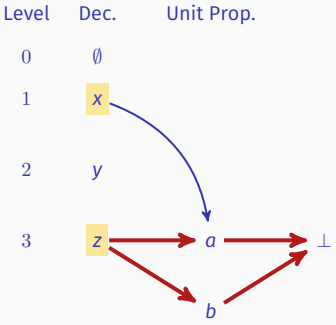
- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current

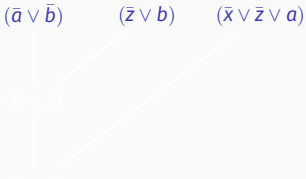
Clause learning



- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

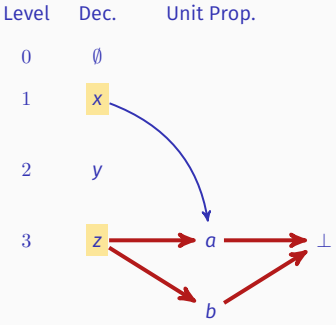
Clause learning

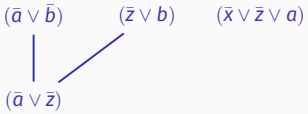




- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

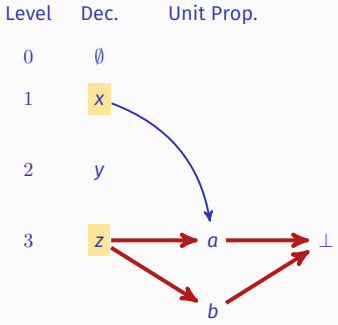
Clause learning

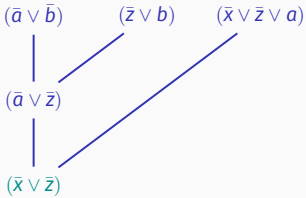




- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

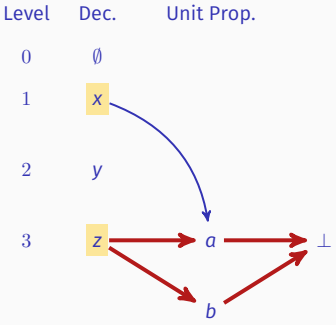
Clause learning

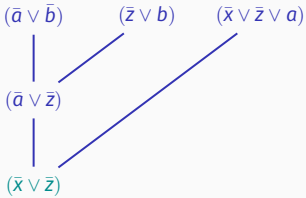




- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

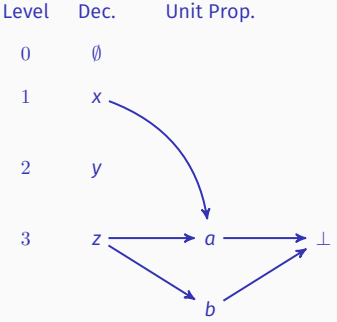
Clause learning



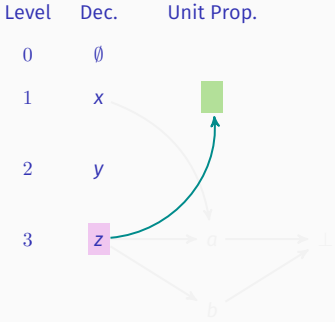


- Analyze conflict [MS95, MSS96a, MSS96a, MSS96b, MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause learning – after backtracking

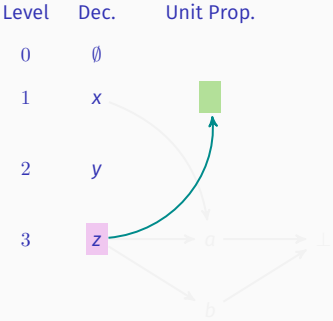


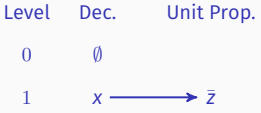
Clause learning – after backtracking



- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

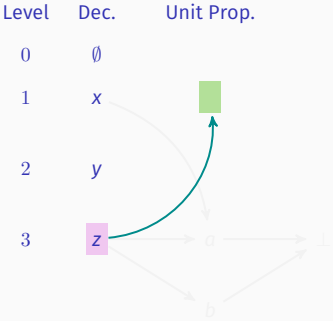
Clause learning – after backtracking

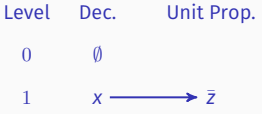




- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

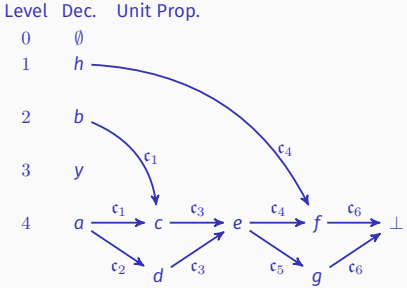
Clause learning – after backtracking



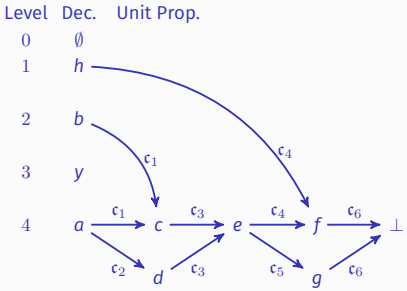


- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are asserting (with exceptions) [MS95, MSS96b, MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [MMZ+01]

Quiz – conflict analysis

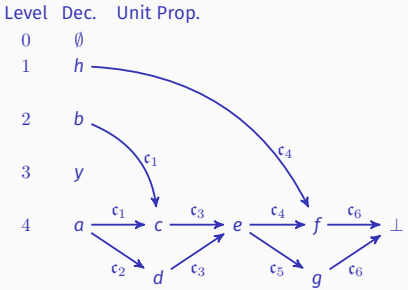


Quiz – conflict analysis



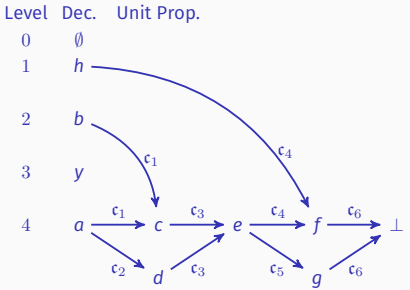
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$

Quiz – conflict analysis



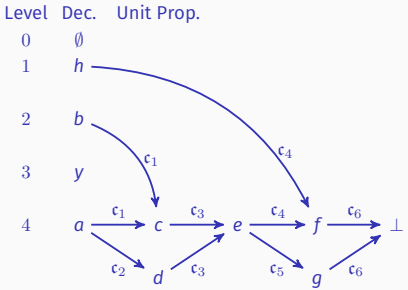
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$

Quiz – conflict analysis



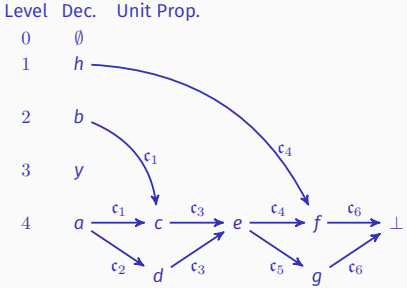
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$

Quiz – conflict analysis



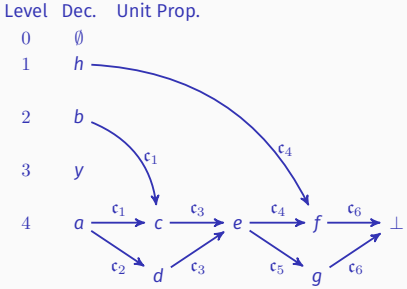
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}\}$	$\{c, d\}$

Quiz – conflict analysis



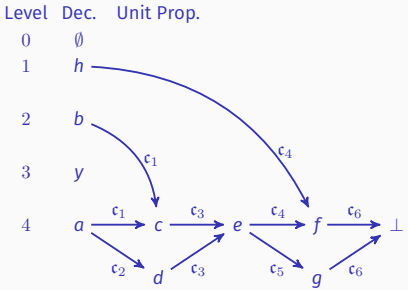
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}\}$	$\{c, d\}$
4	$[c, d]$	c	c_1	$\{\bar{h}, \bar{b}\}$	$\{a\}$

Quiz – conflict analysis



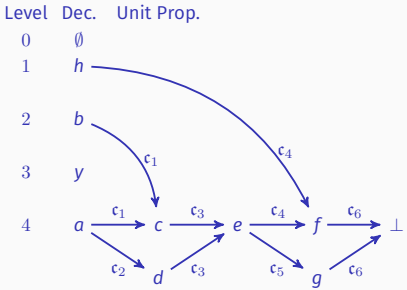
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}\}$	$\{c, d\}$
4	$[c, d]$	c	c_1	$\{\bar{h}, \bar{b}\}$	$\{a\}$
5	$[d, a]$	d	c_2	$\{\bar{h}, \bar{b}\}$	$\emptyset$

Quiz – conflict analysis



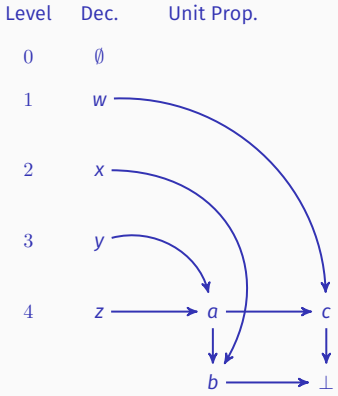
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}\}$	$\{c, d\}$
4	$[c, d]$	c	c_1	$\{\bar{h}, \bar{b}\}$	$\{a\}$
5	$[d, a]$	d	c_2	$\{\bar{h}, \bar{b}\}$	$\emptyset$
6	$[a]$	a	dec var	$\{\bar{h}, \bar{b}, \bar{a}\}$	–

Quiz – conflict analysis

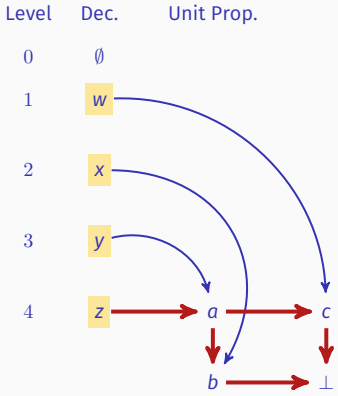


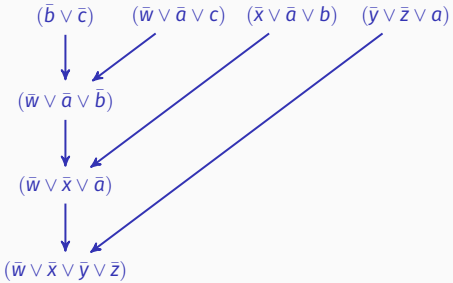
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}\}$	$\{c, d\}$
4	$[c, d]$	c	c_1	$\{\bar{h}, \bar{b}\}$	$\{a\}$
5	$[d, a]$	d	c_2	$\{\bar{h}, \bar{b}\}$	$\emptyset$
6	$[a]$	a	dec var	$\{\bar{h}, \bar{b}, \bar{a}\}$	–
7	$[]$	–	–	$\{\bar{h}, \bar{b}, \bar{a}\}$	–

Unique implication points (UIPs)



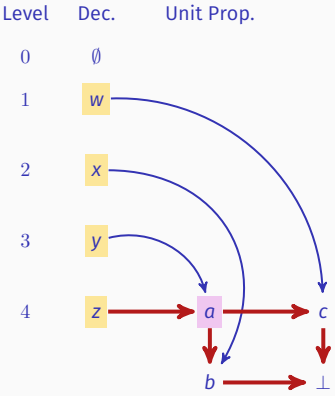
Unique implication points (UIPs)

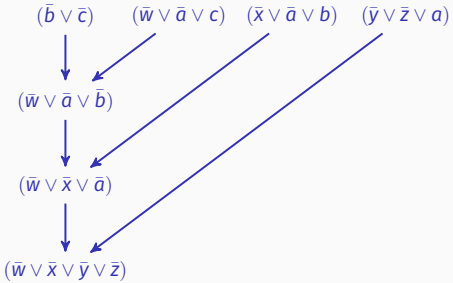




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

Unique implication points (UIPs)

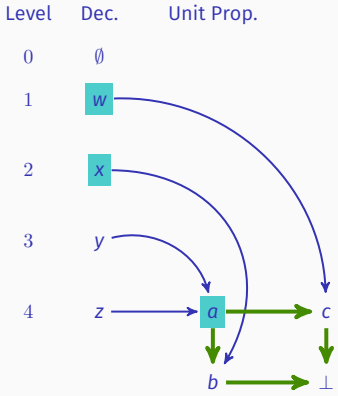


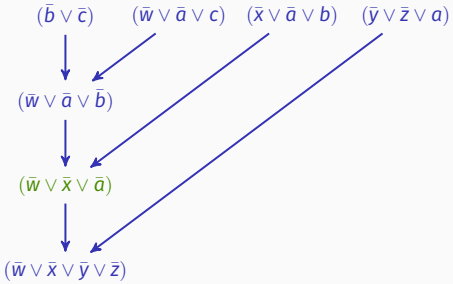


- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP
  - Dominator in DAG for decision level 4

[MS95, MSS99]

Unique implication points (UIPs)

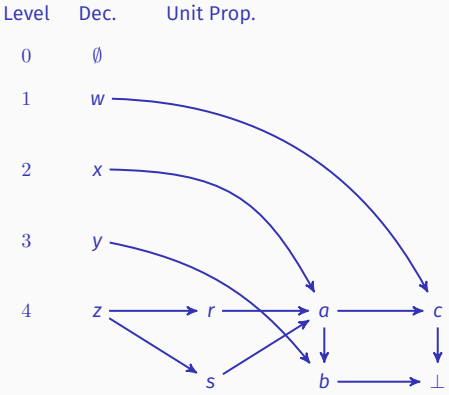




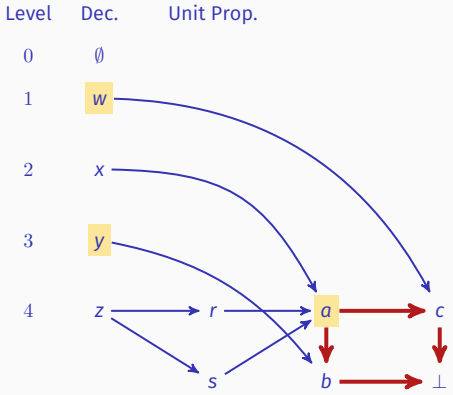
- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
  - Dominator in DAG for level 4
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

[MS95, MSS99]

Multiple UIPs

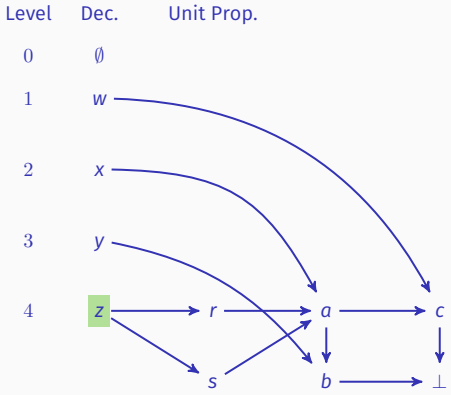


Multiple UIPs



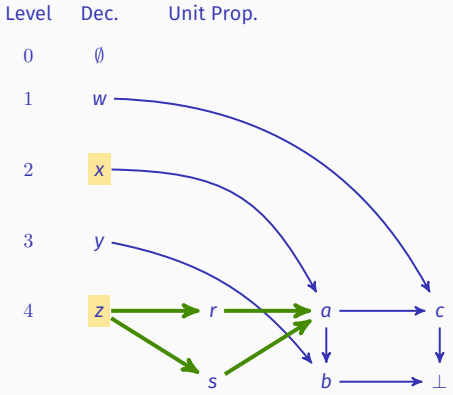
- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$

Multiple UIPs



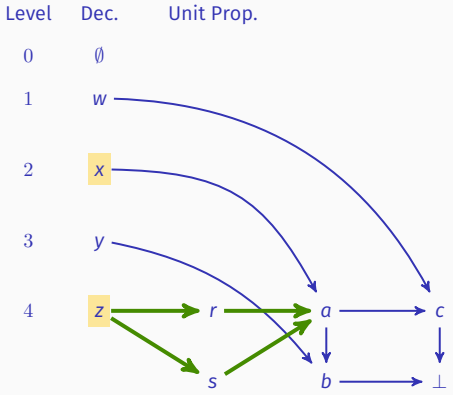
- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$
- But there can be more than 1 UIP

Multiple UIPs



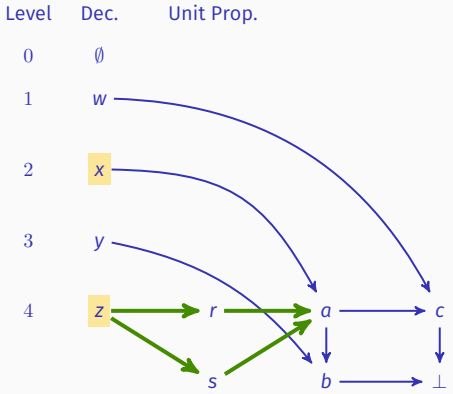
- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\overline{x} \vee \overline{z} \vee a)$
  - Clause is not asserting

Multiple UIPs



- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\overline{x} \vee \overline{z} \vee a)$
  - Clause is not asserting
- In practice smaller clauses more effective
  - Compare with $(\overline{w} \vee \overline{x} \vee \overline{y} \vee \overline{z})$

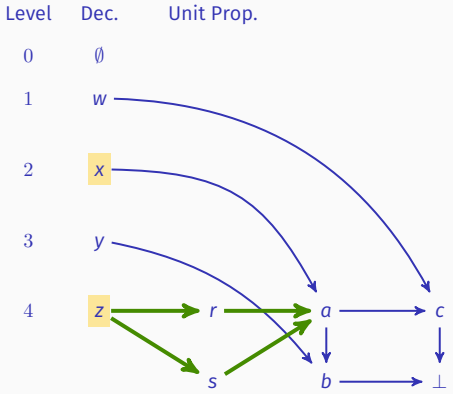
Multiple UIPs



- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\overline{x} \vee \overline{z} \vee a)$
  - Clause is not asserting
- In practice smaller clauses more effective
  - Compare with $(\overline{w} \vee \overline{x} \vee \overline{y} \vee \overline{z})$

- Multiple UIPs proposed in GRASP [MS95, MSS99]
  - First UIP learning proposed in Chaff [MMZ+01]
- Not used in recent state of the art CDCL SAT solvers

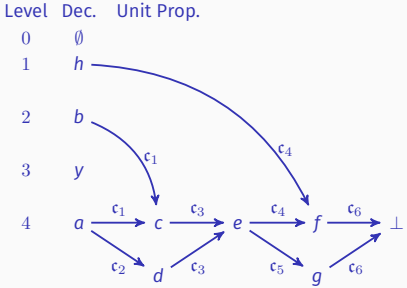
Multiple UIPs



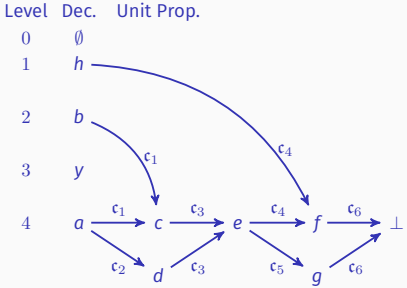
- First UIP:
  - Learn clause $(\overline{w} \vee \overline{y} \vee \overline{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\overline{x} \vee \overline{z} \vee a)$
  - Clause is not asserting
- In practice smaller clauses more effective
  - Compare with $(\overline{w} \vee \overline{x} \vee \overline{y} \vee \overline{z})$

- Multiple UIPs proposed in GRASP [MS95, MSS99]
  - First UIP learning proposed in Chaff [MMZ+01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on some instances [SSS12]

Quiz – conflict analysis with UIP(s)

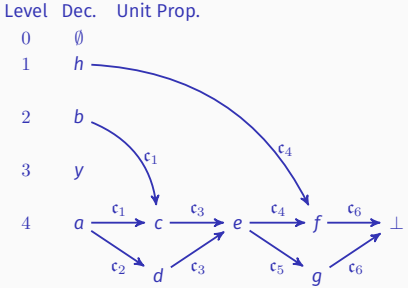


Quiz – conflict analysis with UIP(s)



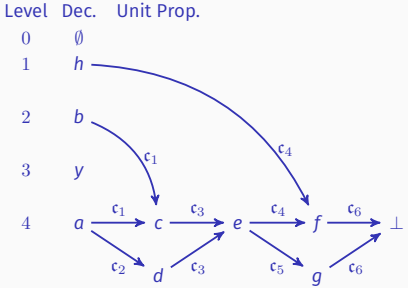
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$

Quiz – conflict analysis with UIP(s)



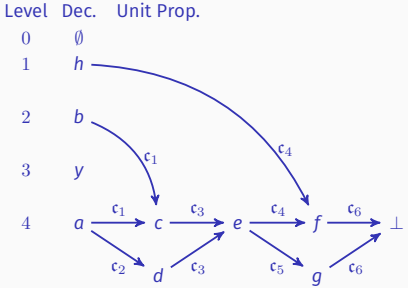
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$

Quiz – conflict analysis with UIP(s)



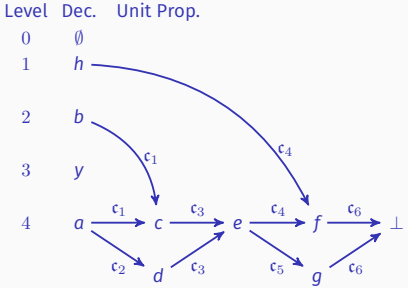
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$

Quiz – conflict analysis with UIP(s)



Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}, \bar{e}\}$	$\emptyset$

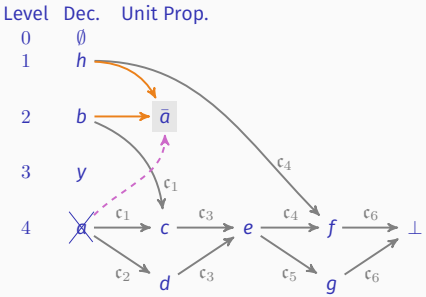
Quiz – conflict analysis with UIP(s)



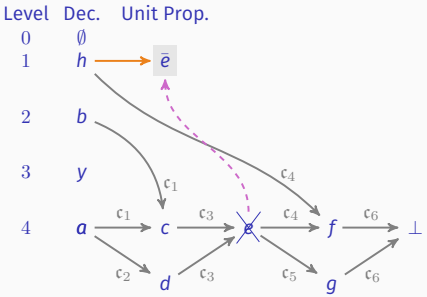
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	c_6	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	c_4	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	c_5	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	c_3	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	$[]$	–	–	$\{\bar{h}, \bar{e}\}$	–

Quiz (Cont.) – non-chronological backtracking

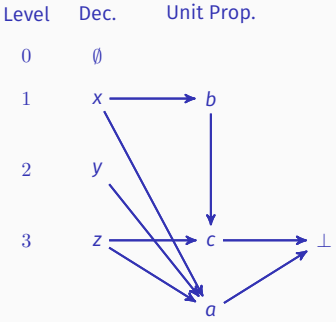
Without UIP:



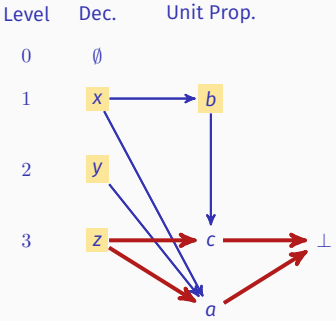
With UIP:

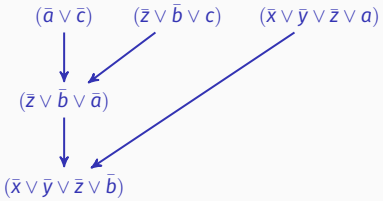


Clause minimization I



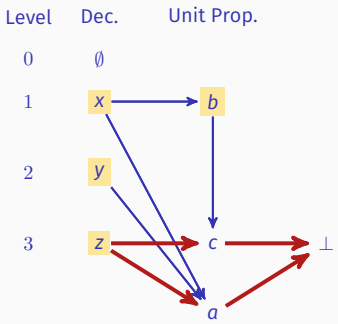
Clause minimization I

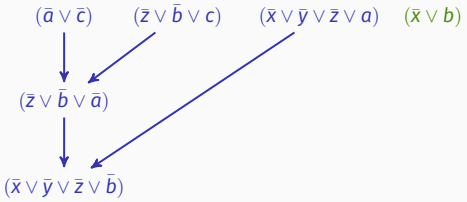




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

Clause minimization I

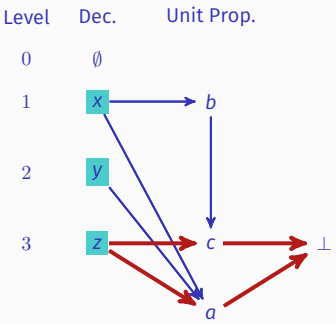


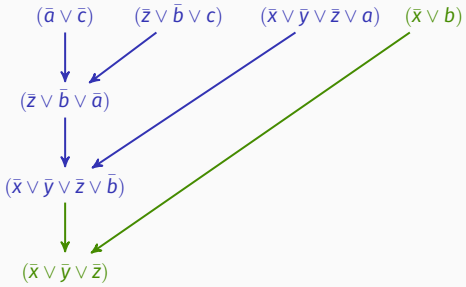


- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization)

[SB09]

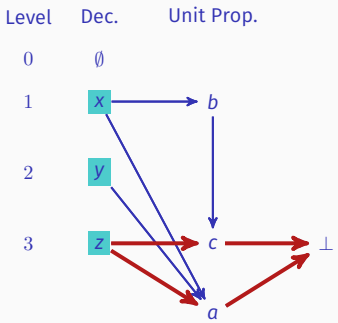
Clause minimization I

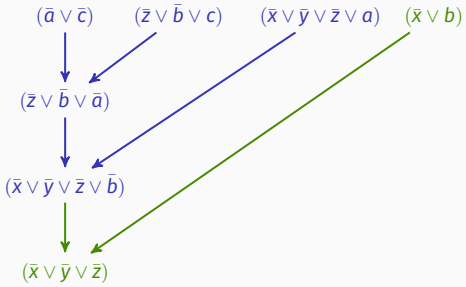




- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

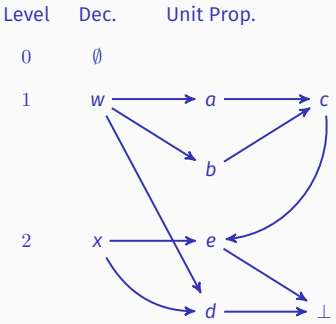
Clause minimization I



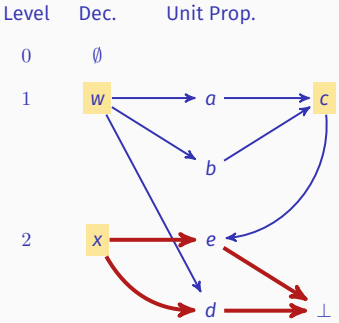


- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization) [SB09]
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

Clause minimization II

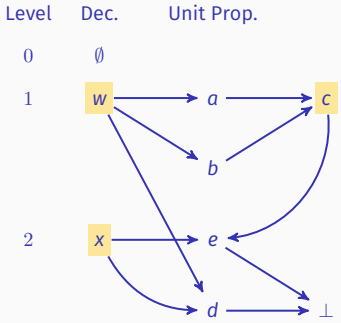


Clause minimization II



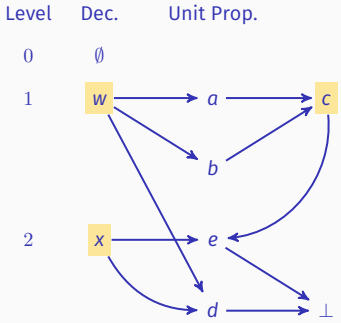
- Learn clause $(\overline{w} \vee \bar{x} \vee \bar{c})$

Clause minimization II



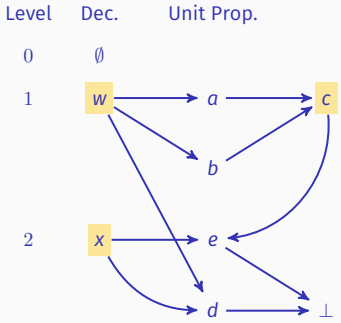
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

Clause minimization II



- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

Clause minimization II

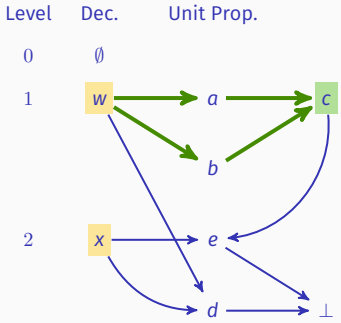


- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

- Marked nodes: literals in learned clause

[SB09]

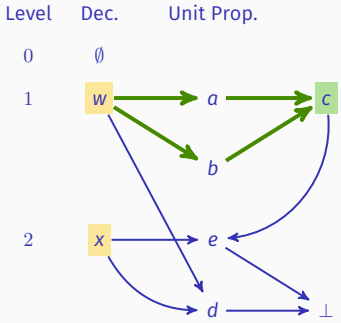
Clause minimization II



- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new decision nodes
  - Drop literal c if only marked nodes visited

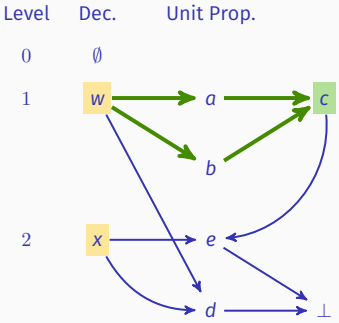
Clause minimization II



- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization
- Learn clause $(\bar{w} \vee \bar{x})$

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new decision nodes
  - Drop literal c if only marked nodes visited

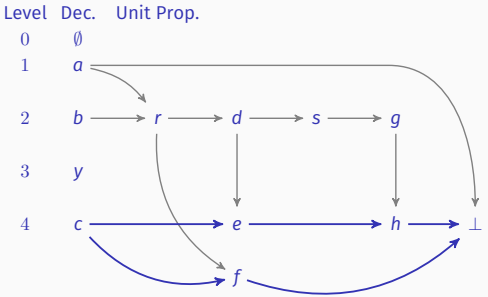
Clause minimization II



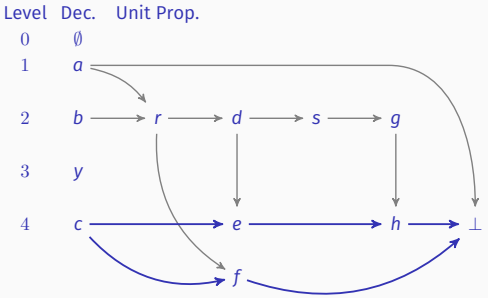
- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$~~
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$
- Can apply recursive minimization
- Learn clause $(\bar{w} \vee \bar{x})$

- Marked nodes: literals in learned clause [SB09]
- Trace back from c until marked nodes or new decision nodes
  - Drop literal c if only marked nodes visited
- Recursive minimization runs in (amortized) linear time

Quiz – conflict clause minimization



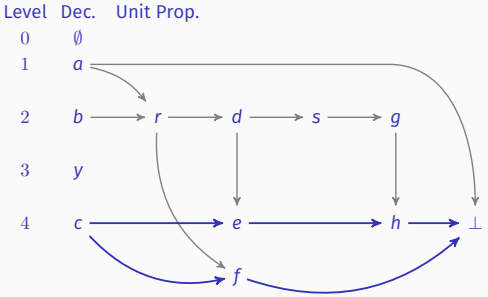
Quiz – conflict clause minimization



Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Quiz – conflict clause minimization

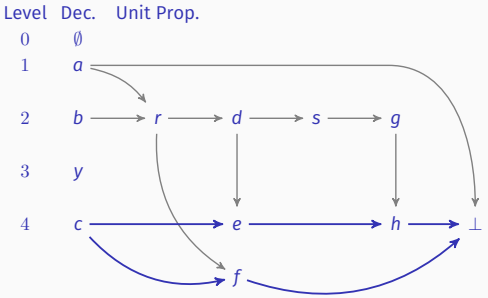


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action

Quiz – conflict clause minimization

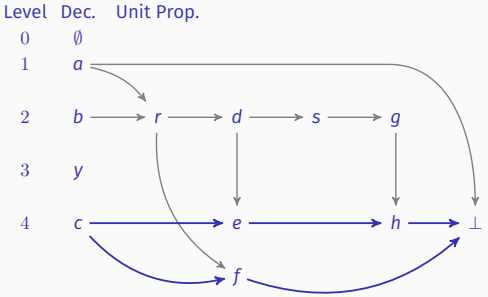


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–

Quiz – conflict clause minimization

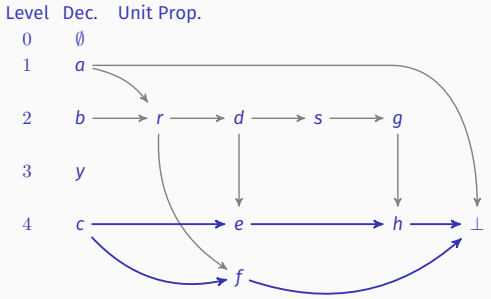


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–

Quiz – conflict clause minimization

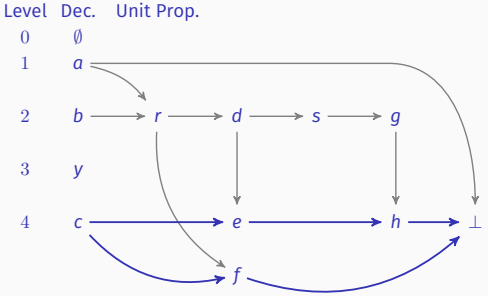


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–
g	d	$\{a, d, r, c\}$	$\emptyset$	$[]$	d marked, skip

Quiz – conflict clause minimization

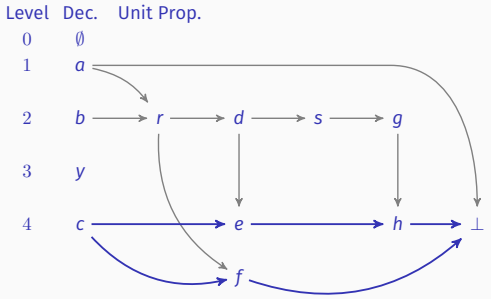


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \quad)$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–
g	d	$\{a, d, r, c\}$	$\emptyset$	$[]$	d marked, skip
g	–	$\{a, d, r, c\}$	$\emptyset$	$[]$	no unmarked vars; $\therefore$ drop g

Quiz – conflict clause minimization

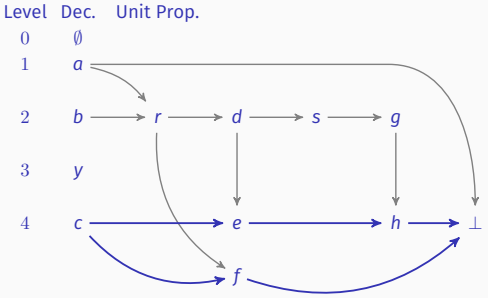


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \quad)$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–
g	d	$\{a, d, r, c\}$	$\emptyset$	$[]$	d marked, skip
g	–	$\{a, d, r, c\}$	$\emptyset$	$[]$	no unmarked vars; $\therefore$ drop g
d	d	$\{a, r, c\}$	$\emptyset$	$[r]$	–

Quiz – conflict clause minimization

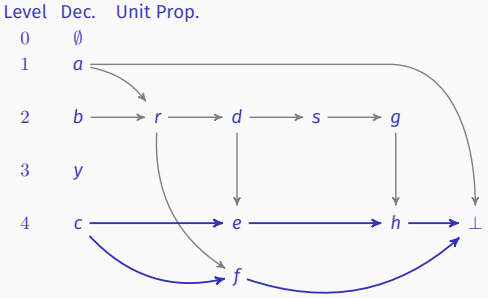


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \quad)$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–
g	d	$\{a, d, r, c\}$	$\emptyset$	$[]$	d marked, skip
g	–	$\{a, d, r, c\}$	$\emptyset$	$[]$	no unmarked vars; $\therefore$ drop g
d	d	$\{a, r, c\}$	$\emptyset$	$[r]$	–
d	r	$\{a, r, c\}$	$\emptyset$	$[]$	r marked, skip

Quiz – conflict clause minimization

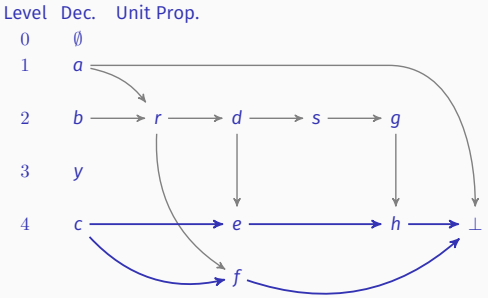


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \quad)$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
g	g	$\{a, d, r, c\}$	$\emptyset$	$[s]$	–
g	s	$\{a, d, r, c\}$	$\emptyset$	$[d]$	–
g	d	$\{a, d, r, c\}$	$\emptyset$	$[]$	d marked, skip
g	–	$\{a, d, r, c\}$	$\emptyset$	$[]$	no unmarked vars; $\therefore$ drop g
d	d	$\{a, r, c\}$	$\emptyset$	$[r]$	–
d	r	$\{a, r, c\}$	$\emptyset$	$[]$	r marked, skip
d	–	$\{a, r, c\}$	$\emptyset$	$[]$	no unmarked vars; $\therefore$ drop d

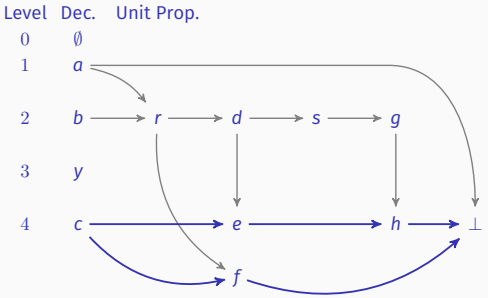
Quiz – conflict clause minimization (cont.)



Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c})$

Quiz – conflict clause minimization (cont.)

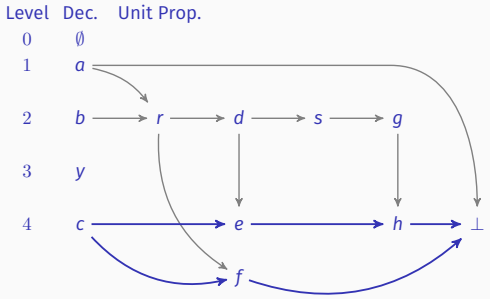


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action

Quiz – conflict clause minimization (cont.)

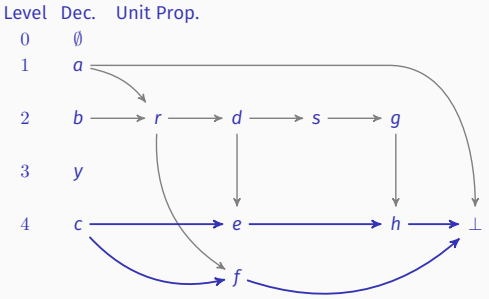


Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
r	r	$\{a, c\}$	$\emptyset$	$[a, b]$	–
r	a	$\{a, c\}$	$\emptyset$	$[b]$	a marked
r	b	$\{a, c\}$	$\{b\}$	$[]$	b decision & unmarked
r	–	$\{a, c\}$	$\{b\}$	$[]$	unmarked vars; $\therefore$ keep r

Quiz – conflict clause minimization (cont.)



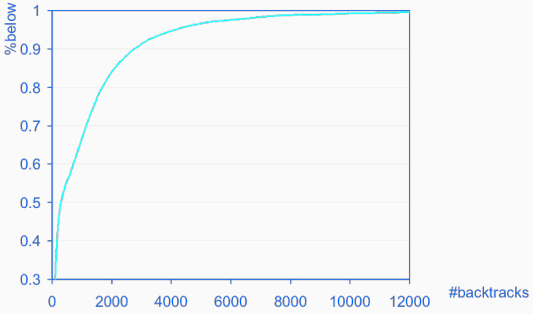
Learned clause: $(\bar{a} \vee \bar{r} \vee \bar{c} \vee \bar{d} \vee \bar{g})$

Minimized clause: $(\bar{a} \vee \bar{r} \vee \bar{c})$

Target	Curr Var	Marked	Unmarked	Vars to Trace	Action
r	r	$\{a, c\}$	$\emptyset$	$[a, b]$	–
r	a	$\{a, c\}$	$\emptyset$	$[b]$	a marked
r	b	$\{a, c\}$	$\{b\}$	$[]$	b decision & unmarked
r	–	$\{a, c\}$	$\{b\}$	$[]$	unmarked vars; $\therefore$ keep r
a, c	–	–	$\emptyset$	$[]$	a, c decision variables; keep both

Search Restarts

Branch randomization

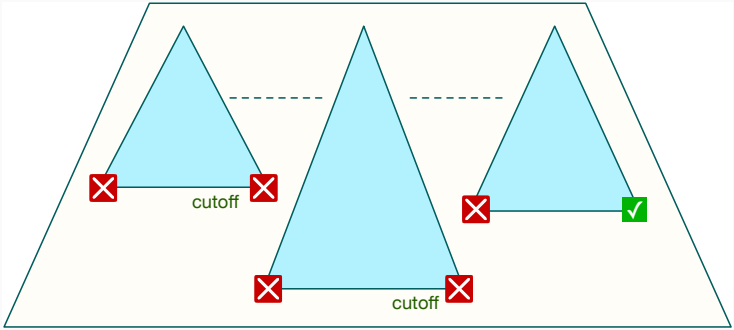


- Heavy-tail behavior: [GSC97]
  - 10000 runs, branching randomization on satisfiable industrial instance

$\therefore$ use rapid randomized restarts (search restarts)

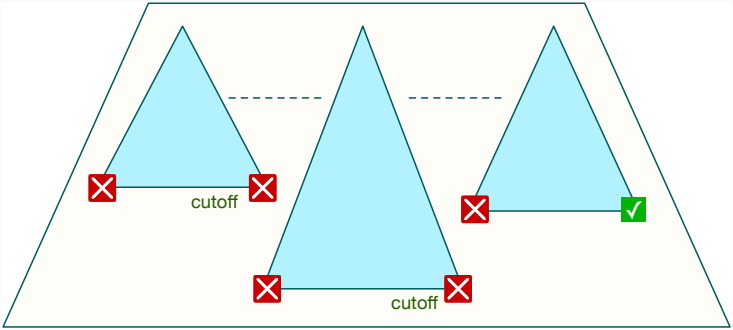
Search restarts

- Restart search after a number of conflicts



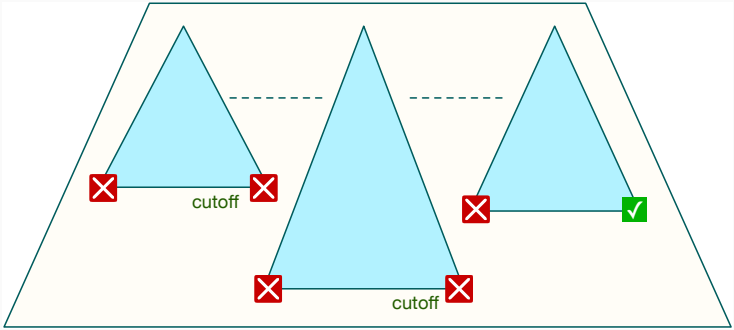
Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist



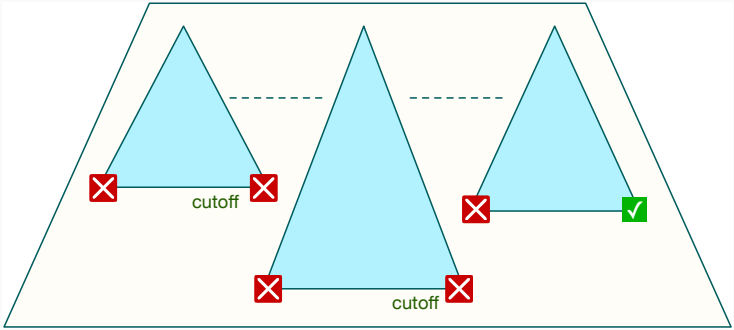
Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist
  - Effective for SAT & UNSAT formulas. **Why?**



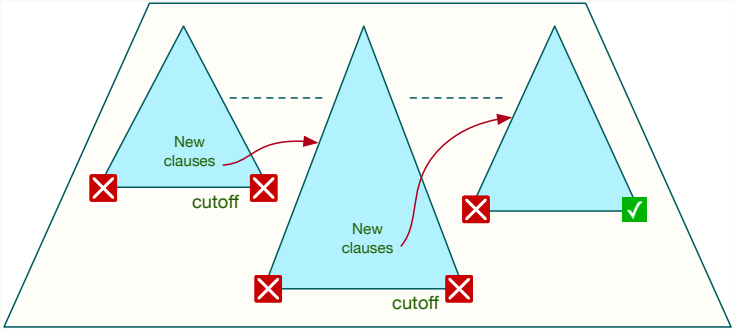
Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist
  - Effective for SAT & UNSAT formulas. **Why?**
    - Proof complexity arguments



Search restarts

- Restart search after a number of conflicts
  - Increase cutoff after each restart
    - Guarantees completeness
    - Different policies exist
  - Effective for SAT & UNSAT formulas. **Why?**
    - Proof complexity arguments
  - Clause learning (very) effective in between restarts



Lazy Data Structures

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why?

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**
- Clause learning to be effective requires a more efficient representation:

Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**
- Clause learning to be effective requires a more efficient representation: **Watched Literals** [MMZ+01]

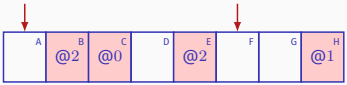
Data structures basics

- Recap states of a clause: unresolved, unit, falsified, satisfied
- Each literal l should access clauses containing l and $\bar{l}$
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals, L
  - Clause learning can generate **large** clauses
    - Worst-case size: $\mathcal{O}(n)$
  - Worst-case number of literals: $\mathcal{O}(m\,n)$
  - In practice,

    Unit propagation slow-down worse than linear as clauses are learned **!**
- Clause learning to be effective requires a more efficient representation: **Watched Literals** [MMZ+01]
  - Watched literals are one example of lazy data structures
    - But there are others [ZS00]

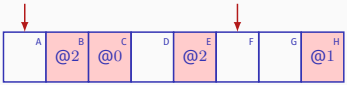
Watched literals

Watched literals



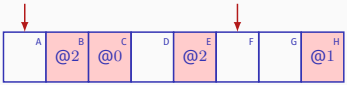
Watch **2** unassigned literals in each clause

Watched literals

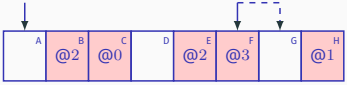


Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

Watched literals

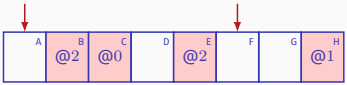


Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

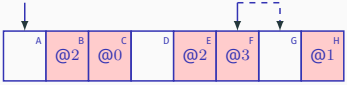


At DLevel 3: watch updated

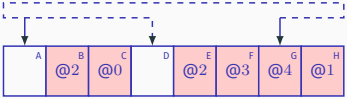
Watched literals



Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved

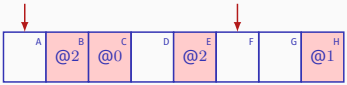


At DLevel 3: watch updated

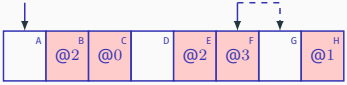


At DLevel 4: watch updated

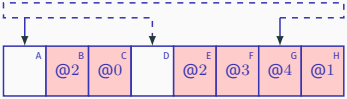
Watched literals



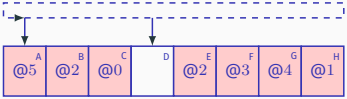
Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved



At DLevel 3: watch updated

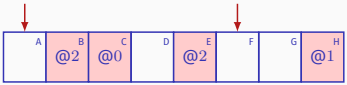


At DLevel 4: watch updated

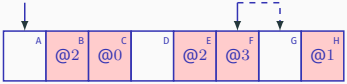


At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied

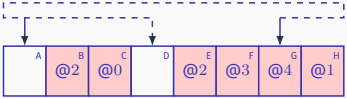
Watched literals



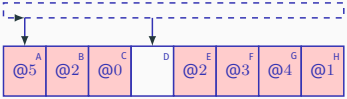
Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved



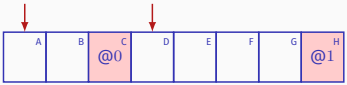
At DLevel 3: watch updated



At DLevel 4: watch updated

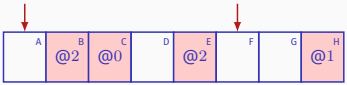


At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied

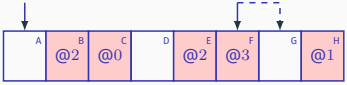


After backtracking to DLevel 1
Watched literals untouched

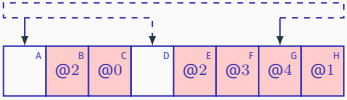
Watched literals – different implementations exist!



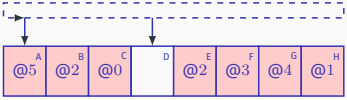
Watch 2 unassigned literals in each clause
At DLevel 2: clause is unresolved



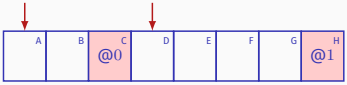
At DLevel 3: watch updated



At DLevel 4: watch updated



At DLevel 5: clause is unit
Literal D assigned value 1; clause becomes satisfied



After backtracking to DLevel 1
Watched literals untouched

Additional key techniques

- Conflict-driven branching [MMZ+01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
  - Recent promising ML-based branching [LGPC16a, LGPC16b]

Additional key techniques

- Conflict-driven branching [MMZ+01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
  - Recent promising ML-based branching [LGPC16a, LGPC16b]
- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete larger clauses [MSS96b, MSS99]
  - Delete less used clauses [GN02, ES03]
  - Delete based on LBD metric [AS09]

Additional key techniques

- Conflict-driven branching [MMZ+01]
  - Use conflict to bias variables to branch on, associate score with each variable
  - Prefer recent bias by regularly decreasing variable scores
  - Recent promising ML-based branching [LGPC16a, LGPC16b]
- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete larger clauses [MSS96b, MSS99]
  - Delete less used clauses [GN02, ES03]
  - Delete based on LBD metric [AS09]
- Other effective techniques:
  - Phase saving [PD07]
  - Novel restart strategies [Hua07, BF15, LOM+18]
  - Preprocessing/inprocessing [JHB12, HJL+15]
  - Clause minimization: LBD-based and UP-based [AS09, LLX+17]

Why CDCL Works?

Why CDCL works – a practitioner's view

- GRASP-like clause learning extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit's internal structure
    - Several ideas originated in earlier work [MSS93, MSS94]

Why CDCL works – a practitioner's view

- GRASP-like clause learning extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit's internal structure
    - Several ideas originated in earlier work [MSS93, MSS94]

- Understanding problem structure is **essential**
  - Clauses are learned locally to each decision level
  - UIPs further localize the learned clauses
  - GRASP-like clause learning aims at learning small clauses, related with the sources of conflicts
  - Most practical problem instances exhibit the structure GRASP-like clause learning is most effective on
    - Most problems are **not** natively represented in clausal form [Stu13]

Why CDCL works – a practitioner's view

- GRASP-like clause learning extensively inspired in circuit reasoners
  - UIPs mimic unique sensitization points (USPs), from testing
  - Analysis of conflicts organized by decision levels
    - In circuits, branching is (mostly) on the inputs, e.g. PODEM, FAN, etc.
    - Need to find ways to exploit the circuit's internal structure
    - Several ideas originated in earlier work [MSS93, MSS94]
- Understanding problem structure is **essential**
  - Clauses are learned locally to each decision level
  - UIPs further localize the learned clauses
  - GRASP-like clause learning aims at learning small clauses, related with the sources of conflicts
  - Most practical problem instances exhibit the structure GRASP-like clause learning is most effective on
    - Most problems are **not** natively represented in clausal form [Stu13]
- There are also proof complexity arguments [BKS04, PD09, PD11]

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]
  - Use activation/selector/indicator variables

Given clause	Added to SAT solver
c_i	$c_i \vee \overline{s_i}$

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]
  - Use activation/selector/indicator variables

Given clause	Added to SAT solver
c_i	$c_i \vee \overline{s_i}$

  - To activate clause: add assumption $s_i = 1$

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]
  - Use activation/selector/indicator variables

Given clause	Added to SAT solver
c_i	$c_i \vee \overline{s_i}$

  - To activate clause: add assumption $s_i = 1$
  - To deactivate clause: add assumption $s_i = 0$ (optional)

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]
  - Use activation/selector/indicator variables

Given clause	Added to SAT solver
c_i	$c_i \vee \overline{s_i}$

  - To activate clause: add assumption $s_i = 1$
  - To deactivate clause: add assumption $s_i = 0$ (optional)
  - To remove clause: add unit $(\overline{s_i})$

Incremental SAT solving

- SAT solver often called multiple times on related formulas
- It helps to make incremental changes **&** remember already learned clauses (that still hold)
- Most often used solution: [ES03]
  - Use activation/selector/indicator variables

Given clause	Added to SAT solver
c_i	$c_i \vee \overline{s_i}$

  - To activate clause: add assumption $s_i = 1$
  - To deactivate clause: add assumption $s_i = 0$ (optional)
  - To remove clause: add unit $(\overline{s_i})$
  - **Any** learned clause contains explanation given working assumptions (more next)

An example

$$\mathcal{B} = \{(\bar{a} \vee b), (\bar{a} \vee c)\}$$
$$\mathcal{S} = \{(a \vee \bar{s_1}), (\bar{b} \vee \bar{c} \vee \bar{s_2}), (a \vee \bar{c} \vee \bar{s_3}), (a \vee \bar{b} \vee \bar{s_4})\}$$

- Background knowledge $\mathcal{B}$: final clauses, i.e. no indicator variables
- Soft clauses $\mathcal{S}$: add indicator variables $\{s_1, s_2, s_3, s_4\}$

An example

$$\mathcal{B} = \{(\bar{a} \vee b), (\bar{a} \vee c)\}$$
$$\mathcal{S} = \{(a \vee \overline{s_1}), (\bar{b} \vee \bar{c} \vee \overline{s_2}), (a \vee \bar{c} \vee \overline{s_3}), (a \vee \bar{b} \vee \overline{s_4})\}$$

- Background knowledge $\mathcal{B}$: final clauses, i.e. no indicator variables
- Soft clauses $\mathcal{S}$: add indicator variables $\{s_1, s_2, s_3, s_4\}$
- E.g. given assumptions $\{s_1 = 1, s_2 = 0, s_3 = 0, s_4 = 1\}$, SAT solver handles formula:

$$\mathcal{F} = \{(\bar{a} \vee b), (\bar{a} \vee c), (a), (a \vee \bar{b})\}$$

which is satisfiable

Quiz – what happens in this case?

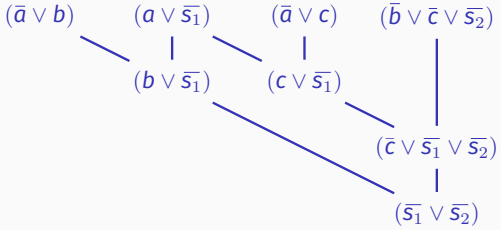
$$\mathcal{B} = \{(\bar{a} \vee b), (\bar{a} \vee c)\}$$
$$\mathcal{S} = \{(a \vee \overline{s_1}), (\bar{b} \vee \bar{c} \vee \overline{s_2}), (a \vee \bar{c} \vee \overline{s_3}), (a \vee \bar{b} \vee \overline{s_4})\}$$

- Given assumptions $\{s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1\}$?

Quiz – what happens in this case?

$$\mathcal{B} = \{(\bar{a} \vee b), (\bar{a} \vee c)\}$$
$$\mathcal{S} = \{(a \vee \overline{s_1}), (\bar{b} \vee \bar{c} \vee \overline{s_2}), (a \vee \bar{c} \vee \overline{s_3}), (a \vee \bar{b} \vee \overline{s_4})\}$$

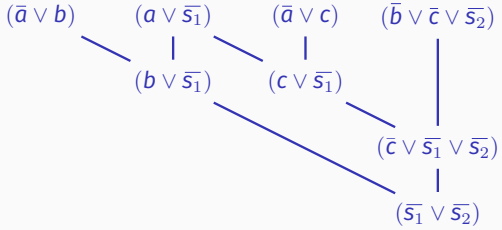
- Given assumptions $\{s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1\}$?



Quiz – what happens in this case?

$$\mathcal{B} = \{(\bar{a} \vee b), (\bar{a} \vee c)\}$$
$$\mathcal{S} = \{(a \vee \bar{s_1}), (\bar{b} \vee \bar{c} \vee \bar{s_2}), (a \vee \bar{c} \vee \bar{s_3}), (a \vee \bar{b} \vee \bar{s_4})\}$$

- Given assumptions $\{s_1 = 1, s_2 = 1, s_3 = 1, s_4 = 1\}$?



- Unsatisfiable core: 1^{st} and 2^{nd} clauses of $\mathcal{S}$, given $\mathcal{B}$

3 Modeling with SAT

Recap Clausification of Boolean Formulas

How to translate to CNF?

How to translate to CNF?

- **Obs:** *There are no CNF formulas* [Stu13]

How to translate to CNF?

- **Obs:** *There are no CNF formulas* [Stu13]

- Standard textbook solution
  - Operator elimination; De Morgan's laws, remove double negations & apply distributivity
  - Worst-case exponential
  - Set of variables constant

How to translate to CNF?

- **Obs:** *There are no CNF formulas* [Stu13]

- Standard textbook solution
  - Operator elimination; De Morgan's laws, remove double negations & apply distributivity
  - Worst-case exponential
  - Set of variables constant

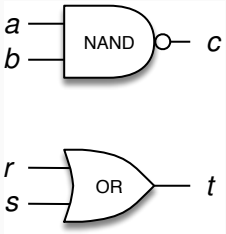
- Tseitin's translation & variants (next)
  - New variables added
  - Satisfiability is preserved
  - Linear size transformation

Representing Boolean formulas / circuits I

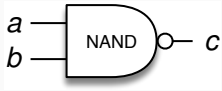
- Satisfiability problems can be defined on Boolean circuits/formulas
  - Can use any logic connective: $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \ldots$
- Can represent circuits/formulas as CNF formulas [Tse68, PG86]
  - For each (simple) gate, CNF formula encodes the consistent assignments to the gate's inputs and output
    - Given $z = \mathrm{OP}(x, y)$, represent in CNF $z \leftrightarrow \mathrm{OP}(x, y)$
  - CNF formula for the circuit is the conjunction of CNF formula for each gate

$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$

$$\mathcal{F}_t = (\bar{r} \vee t) \wedge (\bar{s} \vee t) \wedge (r \vee s \vee \bar{t})$$



Representing Boolean formulas / circuits II



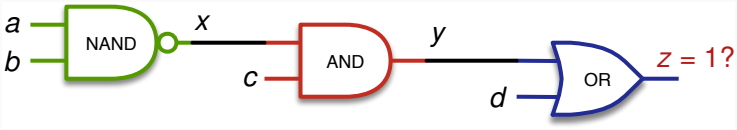
ab / c	00	01	11	10
0	0	0	1	0
1	1	1	0	1

a	b	c	$\mathcal{F}_c$(a,b,c)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$

Representing Boolean formulas / circuits III

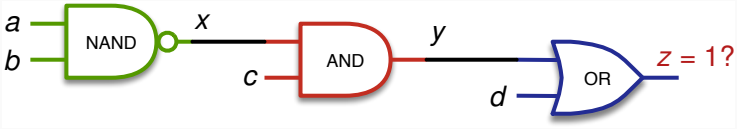
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



$$\begin{aligned}\mathcal{F} \quad = \quad & (a \vee x) \wedge (b \vee x) \wedge (\bar{a} \vee \bar{b} \vee \bar{x}) \wedge \\ & (x \vee \bar{y}) \wedge (c \vee \bar{y}) \wedge (\bar{x} \vee \bar{c} \vee y) \wedge \\ & (\bar{y} \vee z) \wedge (\bar{d} \vee z) \wedge (y \vee d \vee \bar{z}) \wedge (z)\end{aligned}$$

Representing Boolean formulas / circuits III

- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses

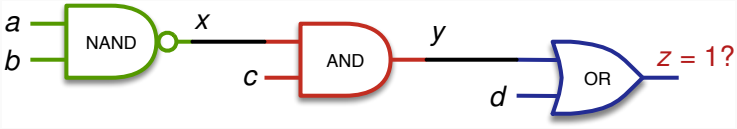


$$\begin{aligned}\mathcal{F} \quad = \quad & (a \vee x) \wedge (b \vee x) \wedge (\bar{a} \vee \bar{b} \vee \bar{x}) \wedge \\ & (x \vee \bar{y}) \wedge (c \vee \bar{y}) \wedge (\bar{x} \vee \bar{c} \vee y) \wedge \\ & (\bar{y} \vee z) \wedge (\bar{d} \vee z) \wedge (y \vee d \vee \bar{z}) \wedge (z)\end{aligned}$$

- Note: $z = d \vee (c \wedge (\neg(a \wedge b)))$
  - **No** distinction between Boolean circuits and (non-clausal) formulas, besides adding new variables

Representing Boolean formulas / circuits III

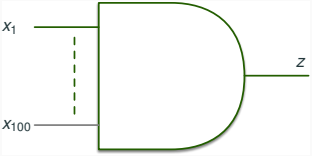
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



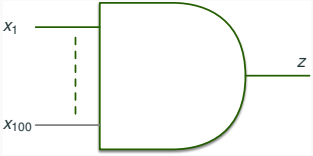
$$\begin{aligned}\mathcal{F} \quad = \quad & (a \vee x) \wedge (b \vee x) \wedge (\bar{a} \vee \bar{b} \vee \bar{x}) \wedge \\ & (x \vee \bar{y}) \wedge (c \vee \bar{y}) \wedge (\bar{x} \vee \bar{c} \vee y) \wedge \\ & (\bar{y} \vee z) \wedge (\bar{d} \vee z) \wedge (y \vee d \vee \bar{z}) \wedge (z)\end{aligned}$$

- Note: $z = d \vee (c \wedge (\neg(a \wedge b)))$
  - **No** distinction between Boolean circuits and (non-clausal) formulas, besides adding new variables
- Easy to do more structures: ITEs; Adders; etc.

Quiz – how to encode a 100 input gate?

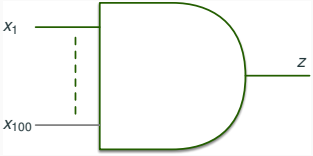


Quiz – how to encode a 100 input gate?



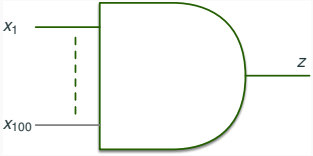
- Impractical to create the truth table...

Quiz – how to encode a 100 input gate?



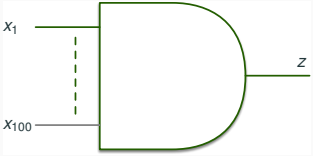
- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$

Quiz – how to encode a 100 input gate?



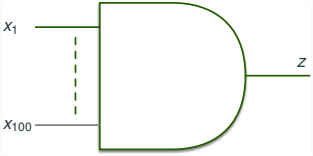
- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$

Quiz – how to encode a 100 input gate?



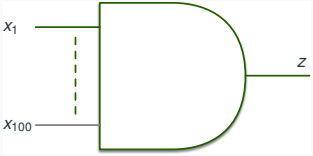
- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all i $x_i = 1$, then $z = 1$

Quiz – how to encode a 100 input gate?



- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all i $x_i = 1$, then $z = 1$, i.e. $\wedge_i x_i \rightarrow z$

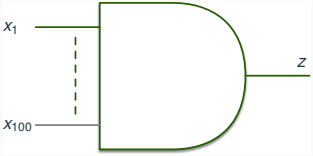
Quiz – how to encode a 100 input gate?



- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all i $x_i = 1$, then $z = 1$, i.e. $\wedge_i x_i \rightarrow z$
- Resulting CNF encoding:

$$\bigwedge_{i=1}^{100} (x_i \vee \bar{z}) \wedge (\overline{x_1} \vee \cdots \vee \overline{x_{100}} \vee z)$$

Quiz – how to encode a 100 input gate?



- Impractical to create the truth table...
- For any x_i, if $x_i = 0$, then $z = 0$, i.e. $\neg x_i \rightarrow \neg z$
- If for all i $x_i = 1$, then $z = 1$, i.e. $\wedge_i x_i \rightarrow z$
- Resulting CNF encoding:

$$\bigwedge_{i=1}^{100} (x_i \vee \bar{z}) \wedge (\overline{x_1} \vee \cdots \vee \overline{x_{100}} \vee z)$$

- Similar ideas apply for other (simple) logical operators: AND, NAND, OR, NOR, etc.

Hard and Soft Constraints

Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied

Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as ∞

Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as ∞

- An example:
  - How to model linear cost function optimization?

$$\begin{array}{ll} \min & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} & \varphi \end{array}$$

Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as ∞

- An example:
  - How to model linear cost function optimization?

$$\begin{array}{ll} \min & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} & \varphi \end{array}$$

  - Hard constraints: φ

Hard vs. soft constraints

- **Hard**: Constraints that **must** be satisfied
- **Soft**: Constraints that we would like to satisfy, if possible
  - Associate a cost (can be unit) with falsifying each soft constraint
  - For a hard constraint, the cost can be viewed as ∞

- An example:
  - How to model linear cost function optimization?

$$\begin{array}{ll} \min & \sum_{j=1}^{n} c_j x_j \\ \text{s.t.} & \varphi \end{array}$$

  - Hard constraints: φ
  - Soft constraints: $(\overline{x_j})$, each with cost c_j

Linear Constraints

Linear constraints

- Cardinality constraints: $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

Linear constraints

- Cardinality constraints: $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- Pseudo-Boolean constraints: $\sum_{j=1}^{n} a_j x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

Linear constraints

- Cardinality constraints: $\sum_{j=1}^{n} x_j \leq k$?
  - How to handle AtMost1 constraints, $\sum_{j=1}^{n} x_j \leq 1$?
  - General form: $\sum_{j=1}^{n} x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- Pseudo-Boolean constraints: $\sum_{j=1}^{n} a_j x_j \bowtie k$, with $\bowtie \in \{<, \leq, =, \geq, >\}$

- If variables are non-Boolean, e.g. with finite domain
  - Need to encode variables (more later)

Equals1, AtLeast1 & AtMost1 constraints

- $\sum_{j=1}^{n} x_j = 1$: encode with $(\sum_{j=1}^{n} x_j \leq 1) \wedge (\sum_{j=1}^{n} x_j \geq 1)$
- $\sum_{j=1}^{n} x_j \geq 1$: encode with $(x_1 \vee x_2 \vee \ldots \vee x_n)$
- $\sum_{j=1}^{n} x_j \leq 1$ encode with:
  - Pairwise encoding
    - Clauses: $\mathcal{O}(n^2)$; No auxiliary variables
  - Sequential counter [Sin05]
    - Clauses: $\mathcal{O}(n)$; Auxiliary variables: $\mathcal{O}(n)$
  - Bitwise encoding [FP01, Pre07]
    - Clauses: $\mathcal{O}(n \log n)$; Auxiliary variables: $\mathcal{O}(\log n)$
  - ...

Pairwise encoding

- How to (propositionally) encode AtMost1 constraint $a + b + c + d \leq 1$?

Pairwise encoding

- How to (propositionally) encode AtMost1 constraint $a + b + c + d \leq 1$?

$$
\begin{aligned}
a \to \bar{b} \wedge \bar{c} \wedge \bar{d} &\implies (\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \\
b \to \bar{c} \wedge \bar{d} \wedge \bar{a} &\implies (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{b} \vee \bar{a}) \\
c \to \bar{d} \wedge \bar{a} \wedge \bar{b} &\implies (\bar{c} \vee \bar{d}) \wedge (\bar{c} \vee \bar{a}) \wedge (\bar{c} \vee \bar{b}) \\
d \to \bar{a} \wedge \bar{b} \wedge \bar{c} &\implies (\bar{d} \vee \bar{a}) \wedge (\bar{d} \vee \bar{b}) \wedge (\bar{d} \vee \bar{c})
\end{aligned}
$$

  - Encoded as: $(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{c} \vee \bar{d})$

Pairwise encoding

- How to (propositionally) encode AtMost1 constraint $a + b + c + d \leq 1$?

$$
\begin{aligned}
a \to \bar{b} \wedge \bar{c} \wedge \bar{d} &\implies (\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \\
b \to \bar{c} \wedge \bar{d} \wedge \bar{a} &\implies (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{b} \vee \bar{a}) \\
c \to \bar{d} \wedge \bar{a} \wedge \bar{b} &\implies (\bar{c} \vee \bar{d}) \wedge (\bar{c} \vee \bar{a}) \wedge (\bar{c} \vee \bar{b}) \\
d \to \bar{a} \wedge \bar{b} \wedge \bar{c} &\implies (\bar{d} \vee \bar{a}) \wedge (\bar{d} \vee \bar{b}) \wedge (\bar{d} \vee \bar{c})
\end{aligned}
$$

  - Encoded as: $(\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{c} \vee \bar{d})$

- With N variables, number of clauses becomes $\frac{n(n-1)}{2}$
  - But no additional variables

Sequential counter encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with sequential counter:

$$(\bar{x}_1 \vee s_1) \wedge (\bar{x}_n \vee \bar{s}_{n-1}) \wedge$$
$$\bigwedge_{1<i<n} ((\bar{x}_i \vee s_i) \wedge (\bar{s}_{i-1} \vee s_i) \wedge (\bar{x}_i \vee \bar{s}_{i-1}))$$

  - If some $x_j = 1$, then all s_i variables must be assigned
    - $s_i = 1$ for $i \geq j$, and so $x_i = 0$ for $i > j$
    - $s_i = 0$ for $i < j$, and so $x_i = 0$ for $i < j$
    - Thus, all other x_i variables must take value 0
  - If all $x_j = 0$, can find consistent assignment to s_i variables
  - $\mathcal{O}(n)$ clauses ; $\mathcal{O}(n)$ auxiliary variables

Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:

- An example: $x_1 + x_2 + x_3 \leq 1$

Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$

- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$
  - Clauses $(\bar{x}_j \vee (v_i \leftrightarrow b_i)) = (\bar{x}_j \vee l_i)$, $i = 0, \ldots, r-1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise

- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

$$(\bar{x}_1 \vee \bar{v}_1) \wedge (\bar{x}_1 \vee \bar{v}_0)$$
$$(\bar{x}_2 \vee \bar{v}_1) \wedge (\bar{x}_2 \vee v_0)$$
$$(\bar{x}_3 \vee v_1) \wedge (\bar{x}_3 \vee \bar{v}_0)$$

Bitwise encoding

- Encode $\sum_{j=1}^{n} x_j \leq 1$ with bitwise encoding:
  - Auxiliary variables $v_0, \ldots, v_{r-1}$; $r = \lceil \log n \rceil$ (with $n > 1$)
  - If $x_j = 1$, then $v_0 \ldots v_{r-1} = b_0 \ldots b_{r-1}$, the binary encoding of $j - 1$

    $x_j \rightarrow (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}) \Leftrightarrow (\bar{x}_j \vee (v_0 = b_0) \wedge \ldots \wedge (v_{r-1} = b_{r-1}))$
  - Clauses $(\bar{x}_j \vee (v_i \leftrightarrow b_i)) = (\bar{x}_j \vee l_i)$, $i = 0, \ldots, r-1$, where
    - $l_i \equiv v_i$, if $b_i = 1$
    - $l_i \equiv \bar{v}_i$, otherwise
  - If $x_j = 1$, assignment to v_i variables must encode $j - 1$
    - For consistency, all other x variables must not take value 1
  - If all $x_j = 0$, any assignment to v_i variables is consistent
  - $\mathcal{O}(n \log n)$ clauses ; $\mathcal{O}(\log n)$ auxiliary variables
- An example: $x_1 + x_2 + x_3 \leq 1$

	$j-1$	$v_1 v_0$
x_1	0	00
x_2	1	01
x_3	2	10

$$(\bar{x}_1 \vee \bar{v}_1) \wedge (\bar{x}_1 \vee \bar{v}_0)$$
$$(\bar{x}_2 \vee \bar{v}_1) \wedge (\bar{x}_2 \vee v_0)$$
$$(\bar{x}_3 \vee v_1) \wedge (\bar{x}_3 \vee \bar{v}_0)$$

General cardinality constraints

- General form: $\sum_{j=1}^{n} x_j \leq k$ (or $\sum_{j=1}^{n} x_j \geq k$)
  - Operational encoding [War98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not ensure arc-consistency
  - Generalized pairwise
    - Clauses: $\mathcal{O}(2^n)$; no auxiliary variables
  - Sequential counters [Sin05]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - BDDs [ES06]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - Sorting networks [Bat68, ES06]
    - Clauses/Variables: $\mathcal{O}(n \log^2 n)$
  - Cardinality Networks: [ANOR09, ANOR11]
    - Clauses/Variables: $\mathcal{O}(n \log^2 k)$
  - Pairwise Cardinality Networks: [CZ10]
  - ...

Generalized pairwise encoding

- General form: $\sum_{j=1}^{n} x_j \leq k$
- Any combination of $k+1$ true variables is disallowed

Generalized pairwise encoding

- General form: $\sum_{j=1}^{n} x_j \leq k$
- Any combination of $k+1$ true variables is disallowed

- Example: $a + b + c + d \leq 2$

Generalized pairwise encoding

- General form: $\sum_{j=1}^{n} x_j \leq k$
- Any combination of $k+1$ true variables is disallowed

- Example: $a + b + c + d \leq 2$

$$
\begin{aligned}
a \wedge b \rightarrow \bar{c} &\implies (\bar{a} \vee \bar{b} \vee \bar{c}) \\
a \wedge b \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{b} \vee \bar{d}) \\
a \wedge c \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{c} \vee \bar{d}) \\
b \wedge c \rightarrow \bar{d} &\implies (\bar{b} \vee \bar{c} \vee \bar{d})
\end{aligned}
$$

  - Encoded as: $(\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee \bar{d}) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee \bar{d})$

Generalized pairwise encoding

- General form: $\sum_{j=1}^{n} x_j \leq k$
- Any combination of $k+1$ true variables is disallowed

- Example: $a + b + c + d \leq 2$

$$
\begin{aligned}
a \wedge b \rightarrow \bar{c} &\implies (\bar{a} \vee \bar{b} \vee \bar{c}) \\
a \wedge b \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{b} \vee \bar{d}) \\
a \wedge c \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{c} \vee \bar{d}) \\
b \wedge c \rightarrow \bar{d} &\implies (\bar{b} \vee \bar{c} \vee \bar{d})
\end{aligned}
$$

  - Encoded as: $(\bar{a} \vee \bar{b} \vee \bar{c}) \wedge (\bar{a} \vee \bar{b} \vee \bar{d}) \wedge (\bar{a} \vee \bar{c} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee \bar{d})$

- In general, number of clauses is C_{k+1}^{n}
  - Recall: for AtMost1 (i.e. for $k = 1$), number of clauses is: $\frac{n(n-1)}{2}$

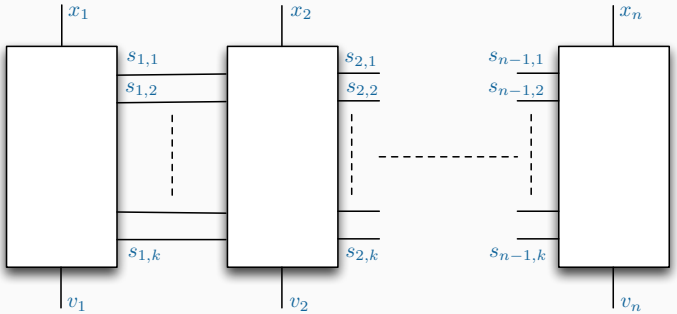
Another example

- Example: $a + b + c + d + e \leq 2$
- Encoding will contain $C_3^5 = 10$ clauses:

$$
\begin{aligned}
a \wedge b \rightarrow \bar{c} &\implies (\bar{a} \vee \bar{b} \vee \bar{c}) \\
a \wedge b \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{b} \vee \bar{d}) \\
a \wedge b \rightarrow \bar{e} &\implies (\bar{a} \vee \bar{b} \vee \bar{e}) \\
a \wedge c \rightarrow \bar{d} &\implies (\bar{a} \vee \bar{c} \vee \bar{d}) \\
a \wedge c \rightarrow \bar{e} &\implies (\bar{a} \vee \bar{c} \vee \bar{e}) \\
a \wedge d \rightarrow \bar{e} &\implies (\bar{a} \vee \bar{d} \vee \bar{e}) \\
b \wedge c \rightarrow \bar{d} &\implies (\bar{b} \vee \bar{c} \vee \bar{d}) \\
b \wedge c \rightarrow \bar{e} &\implies (\bar{b} \vee \bar{c} \vee \bar{e}) \\
b \wedge d \rightarrow \bar{e} &\implies (\bar{b} \vee \bar{d} \vee \bar{e}) \\
c \wedge d \rightarrow \bar{e} &\implies (\bar{c} \vee \bar{d} \vee \bar{e})
\end{aligned}
$$

Sequential counter – revisited I

- Encode $\sum_{j=1}^{n} x_j \leq k$ with sequential counter:



- Equations for each block $1 < i < n$, $1 < j < k$:

$s_i = \sum_{j=1}^{i} x_j$
s_i represented in unary

$s_{i,1} = s_{i-1,1} \vee x_i$
$s_{i,j} = s_{i-1,j} \vee s_{i-1,j-1} \wedge x_i$
$v_i = (s_{i-1,k} \wedge x_i) = 0$

Sequential counter – revisited II

- CNF formula for $\sum_{j=1}^{n} x_j \leq k$:
  - Assume: $k > 0 \land n > 1$
  - Indeces: $1 < i < n\ ,\ 1 < j \leq k$

$$
\begin{aligned}
&(\neg x_1 \lor s_{1,1}) \\
&(\neg s_{1,j}) \\
&(\neg x_i \lor s_{i,1}) \\
&(\neg s_{i-1,1} \lor s_{i,1}) \\
&(\neg x_i \lor \neg s_{i-1,j-1} \lor s_{i,j}) \\
&(\neg s_{i-1,j} \lor s_{i,j}) \\
&(\neg x_i \lor \neg s_{i-1,k}) \\
&(\neg x_n \lor \neg s_{n-1,k})
\end{aligned}
$$

- $\mathcal{O}(n\ k)$ clauses & variables

Pseudo-Boolean constraints

- General form: $\sum_{j=1}^{n} a_j x_j \leq b$
  - Operational encoding [War98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses

Pseudo-Boolean constraints

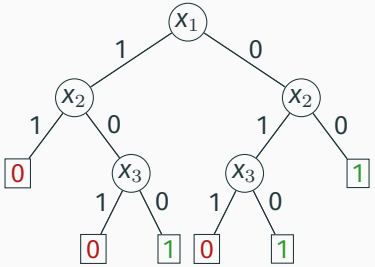
- General form: $\sum_{j=1}^{n} a_j x_j \leq b$
  - Operational encoding [War98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding [BBR09]
    - Let $\nu(n) = \log(n)\log(a_{max})$
    - Clauses: $\mathcal{O}(n^3\nu(n))$; Aux variables: $\mathcal{O}(n^2\nu(n))$

Pseudo-Boolean constraints

- General form: $\sum_{j=1}^{n} a_j x_j \leq b$
  - Operational encoding [War98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding [BBR09]
    - Let $\nu(n) = \log(n)\log(a_{max})$
    - Clauses: $\mathcal{O}(n^3\nu(n))$; Aux variables: $\mathcal{O}(n^2\nu(n))$
  - Improved polynomial watchdog encoding [ANO+12]
    - Clauses & aux variables: $\mathcal{O}(n^3 \log(a_{max}))$
  - ...

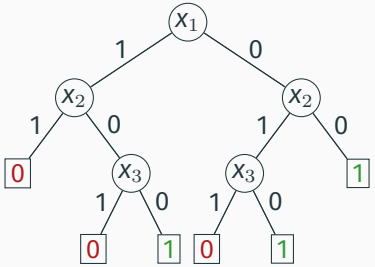
Encoding PB constraints with BDDs I

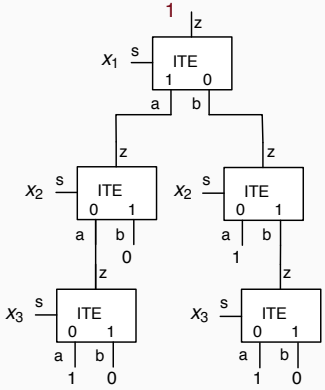
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD



Encoding PB constraints with BDDs I

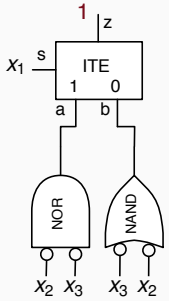
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD





Encoding PB constraints with BDDs II

- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Extract ITE-based circuit from BDD
- Simplify and create final circuit:



More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time (Otherwise P = NP)

[FS02, Tri03, Sel03]

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time (Otherwise P = NP)

[FS02, Tri03, Sel03]

- Example:

$$4x_1 + 3x_2 + 2x_3 = 5$$

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time (Otherwise P = NP)

[FS02, Tri03, Sel03]

- Example:

$$4x_1 + 3x_2 + 2x_3 = 5$$

  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \wedge (4x_1 + 3x_2 + 2x_3 \leq 5)$

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time (Otherwise P = NP)

[FS02, Tri03, Sel03]

- Example:

$$4x_1 + 3x_2 + 2x_3 = 5$$

  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \wedge (4x_1 + 3x_2 + 2x_3 \leq 5)$
  - Let $x_2 = 0$

More on PB constraints

- How about $\sum_{j=1}^{n} a_j x_j = k$?
  - Can use $(\sum_{j=1}^{n} a_j x_j \geq k) \wedge (\sum_{j=1}^{n} a_j x_j \leq k)$, but...
    - $\sum_{j=1}^{n} a_j x_j = k$ is a knapsack constraint
    - Cannot find all consequences in polynomial time (Otherwise P = NP)

[FS02, Tri03, Sel03]

- Example:

$$4x_1 + 3x_2 + 2x_3 = 5$$

  - Replace by $(4x_1 + 3x_2 + 2x_3 \geq 5) \wedge (4x_1 + 3x_2 + 2x_3 \leq 5)$
  - Let $x_2 = 0$
  - Either constraint can still be satisfied, but **not** both

Encoding CSPs

CSP constraints

- Many possible encodings:
  - Direct encoding [dK89, GJ96, Wal00]
  - Log encoding [Wal00]
  - Support encoding [Kas90, Gen02]
  - Log-Support encoding [Gav07]
  - Order encoding for finite linear CSPs [TTKB09]

Direct encoding for CSP w/ binary constraints

- Variable x_i with domain D_i, with $m_i = |D_i|$
- Constraints are relations over domains of variables
  - For a constraint over $x_1, \ldots, x_k$, define relation $R \subseteq D_1 \times \cdots \times D_k$
  - Need to encode elements not in the relation
  - For a binary relation, use set of binary clauses, one for each element not in R
- Represent values of x_i with Boolean variables $x_{i,1}, \ldots, x_{i,m_i}$
- Require $\sum_{k=1}^{m_i} x_{i,k} = 1$
  - Suffices to require $\sum_{k=1}^{m_i} x_{i,k} \geq 1$ [Wal00]
- If the pair of assignments $x_i = v_i \wedge x_j = v_j$ is not allowed, add binary clause $(\bar{x}_{i,v_i} \vee \bar{x}_{j,v_j})$

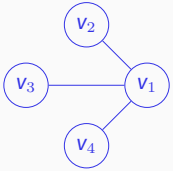
Additional topics

- Encoding problems to SAT is ubiquitous:
  - Many more encodings of finite domain CSP into SAT
  - Encodings of Answer Set Programming (ASP) into SAT
  - Eager SMT solving
  - Theorem provers iteratively encode problems into SAT
  - Model finders interatively encode problems into SAT
  - ...

Modeling Examples

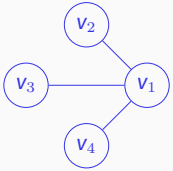
Minimum vertex cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover U of minimum size



Minimum vertex cover

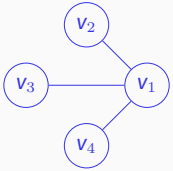
- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover U of minimum size



Vertex cover: $\{v_2, v_3, v_4\}$

Minimum vertex cover

- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover U of minimum size



Vertex cover: $\{v_2, v_3, v_4\}$

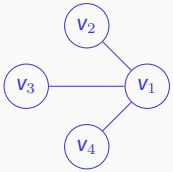
Min vertex cover: $\{v_1\}$

Minimum vertex cover

- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: x_i for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Clauses: $(x_i \vee x_j)$ for each $(v_i, v_j) \in E$
  - Objective function: minimize number of true x_i variables
    - I.e. minimize vertices included in U

Minimum vertex cover

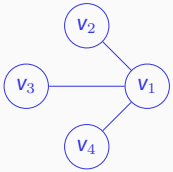
- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: x_i for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Clauses: $(x_i \vee x_j)$ for each $(v_i, v_j) \in E$
  - Objective function: minimize number of true x_i variables
    - I.e. minimize vertices included in U



$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{subject to} & (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \end{array}$$

Minimum vertex cover

- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: x_i for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Clauses: $(x_i \vee x_j)$ for each $(v_i, v_j) \in E$
  - Objective function: minimize number of true x_i variables
    - I.e. minimize vertices included in U



$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{subject to} & (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_1 \vee x_4) \end{array}$$

- Alternative propositional encoding:

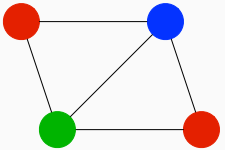
$$\begin{aligned} \varphi_S &= \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_4)\} \\ \varphi_H &= \{(x_1 \vee x_2), (x_1 \vee x_3), (x_1 \vee x_4)\} \end{aligned}$$

Graph coloring

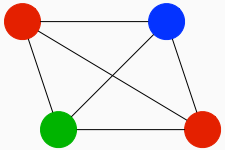
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?

Graph coloring

- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



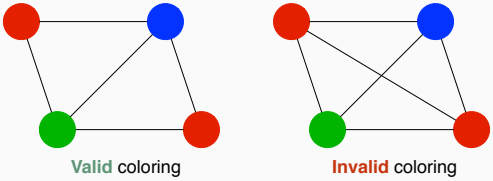
Valid coloring



Invalid coloring

Graph coloring

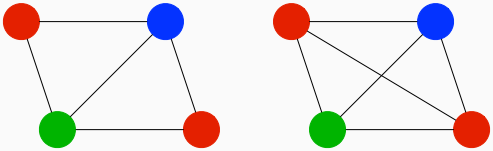
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?

Graph coloring

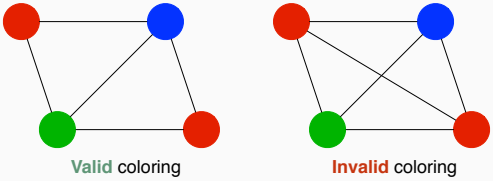
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$

Graph coloring

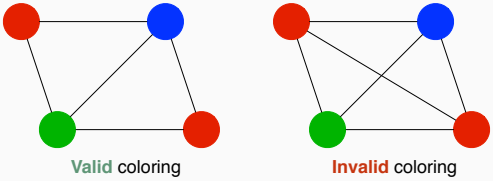
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$
- How to model adjacent vertices with different colors?

Graph coloring

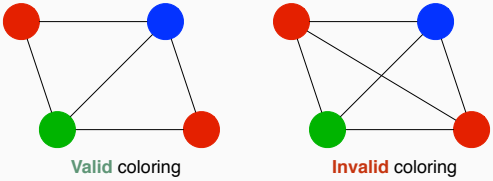
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$
- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \vee \neg x_{l,j})$ **if** $(v_i, v_l) \in E$**, with** $j \in \{1, \ldots, k\}$

Graph coloring

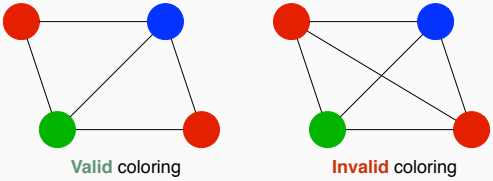
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$
- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \vee \neg x_{l,j})$ **if** $(v_i, v_l) \in E$**, with** $j \in \{1, \ldots, k\}$
- How to model vertices get some color?

Graph coloring

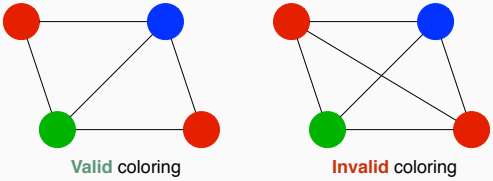
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$
- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \vee \neg x_{l,j})$ **if** $(v_i, v_l) \in E$**, with** $j \in \{1, \ldots, k\}$
- How to model vertices get some color?
  - $\sum_{j \in \{1,\ldots,k\}} x_{i,j} = 1$**, for** $v_i \in V$

Graph coloring

- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?

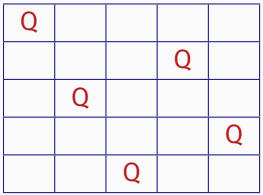


- How to model color assignments to vertices?
  - $x_{i,j} = 1$ **iff vertex** $v_i \in V$ **is assigned color** $j \in \{1, \ldots, k\}$
- How to model adjacent vertices with different colors?
  - $(\neg x_{i,j} \vee \neg x_{l,j})$ **if** $(v_i, v_l) \in E$**, with** $j \in \{1, \ldots, k\}$
- How to model vertices get some color?
  - $\sum_{j \in \{1,\ldots,k\}} x_{i,j} = 1$**, for** $v_i \in V$
  - **Note: it suffices to use** $\left(\bigvee_{j \in \{1,\ldots,k\}} x_{i,j}\right)$

The N-Queens problem I

- The N-Queens Problem:
  Place N queens on a $N \times N$ board, such that no two queens attack each other

- Example for a 5×5 board:



The N-Queens problem II

- x_{ij}: 1 if queen placed in position (i, j); 0 otherwise
- Each row must have exactly one queen:

$$1 \leq i \leq N, \qquad \sum_{j=1}^{N} x_{ij} = 1$$

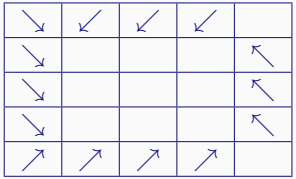
- Each column must have exactly one queen:

$$1 \leq j \leq N, \qquad \sum_{i=1}^{N} x_{ij} = 1$$

- Also, need to define constraints on diagonals...

The N-Queens problem III

- Each diagonal can have at most one queen:



$$i = 1, \qquad 2 \leq j < N, \qquad \sum_{k=0}^{j-1} x_{i+k\, j-k} \leq 1$$

$$i = N, \qquad 1 \leq j < N, \qquad \sum_{k=0}^{N-j} x_{i-k\, j+k} \leq 1$$

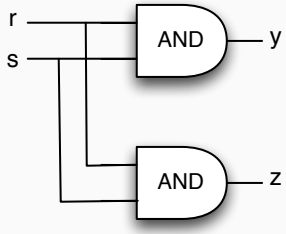
$$j = 1, \qquad 1 \leq i < N, \qquad \sum_{k=0}^{N-i} x_{i+k\, j+k} \leq 1$$

$$j = N, \qquad 2 \leq i < N, \qquad \sum_{k=0}^{i-1} x_{i-k\, j-k} \leq 1$$

Design debugging

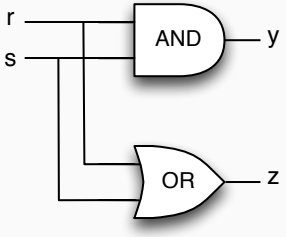
[SMV+07]

Correct circuit



Input stimuli: $\langle r, s \rangle = \langle 0, 1 \rangle$
Valid output: $\langle y, z \rangle = \langle 0, 0 \rangle$

Faulty circuit



Input stimuli: $\langle r, s \rangle = \langle 0, 1 \rangle$
Invalid output: $\langle y, z \rangle = \langle 0, 0 \rangle$

- The model:
  - Hard clauses: Input and output values
  - Soft clauses: CNF representation of circuit
- The problem:
  - Maximize number of satisfied clauses (i.e. circuit gates)

Software package upgrades

[MBC+06, TSJL07, AL08, ALS09, ABL+10b]

- Universe of software packages: $\{p_1, \ldots, p_n\}$
- Associate x_i with p_i: $x_i = 1$ iff p_i is installed
- Constraints associated with package p_i: (p_i, D_i, C_i)
  - D_i: dependencies (required packages) for installing p_i
  - C_i: conflicts (disallowed packages) for installing p_i
- Example problem: Maximum Installability
  - Maximum number of packages that can be installed
  - Package constraints represent hard clauses
  - Soft clauses: (x_i)

Package constraints:

$(p_1, \{p_2 \vee p_3\}, \{p_4\})$
$(p_2, \{p_3\}, \{p_4\})$
$(p_3, \{p_2\}, \emptyset)$
$(p_4, \{p_2, p_3\}, \emptyset)$

Software package upgrades

[MBC+06, TSJL07, AL08, ALS09, ABL+10b]

- Universe of software packages: $\{p_1, \ldots, p_n\}$
- Associate x_i with p_i: $x_i = 1$ iff p_i is installed
- Constraints associated with package p_i: (p_i, D_i, C_i)
  - D_i: dependencies (required packages) for installing p_i
  - C_i: conflicts (disallowed packages) for installing p_i
- Example problem: Maximum Installability
  - Maximum number of packages that can be installed
  - Package constraints represent hard clauses
  - Soft clauses: (x_i)

Package constraints:

$(p_1, \{p_2 \vee p_3\}, \{p_4\})$
$(p_2, \{p_3\}, \{p_4\})$
$(p_3, \{p_2\}, \emptyset)$
$(p_4, \{p_2, p_3\}, \emptyset)$

MaxSAT formulation:

$$\begin{aligned} \varphi_H &= \{(\neg x_1 \vee x_2 \vee x_3), (\neg x_1 \vee \neg x_4), \\ &\quad (\neg x_2 \vee x_3), (\neg x_2 \vee \neg x_4), (\neg x_3 \vee x_2), \\ &\quad (\neg x_4 \vee x_2), (\neg x_4 \vee x_3)\} \\ \varphi_S &= \{(x_1), (x_2), (x_3), (x_4)\} \end{aligned}$$

The knapsack problem

- Given list of pairs (v_i, w_i), $i = 1, \ldots, n$
  - Each pair (v_i, w_i), represents the value and weight of object i

The knapsack problem

- Given list of pairs (v_i, w_i), $i = 1, \ldots, n$
  - Each pair (v_i, w_i), represents the value and weight of object i
- Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed W

The knapsack problem

- Given list of pairs (v_i, w_i), $i = 1, \ldots, n$
  - Each pair (v_i, w_i), represents the value and weight of object i
- Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed W

- Propositional encoding for the knapsack problem?

The knapsack problem

- Given list of pairs (v_i, w_i), $i = 1, \ldots, n$
  - Each pair (v_i, w_i), represents the value and weight of object i
- Pick subset of objects with the maximum sum of values, such that the sum of weights does not exceed W

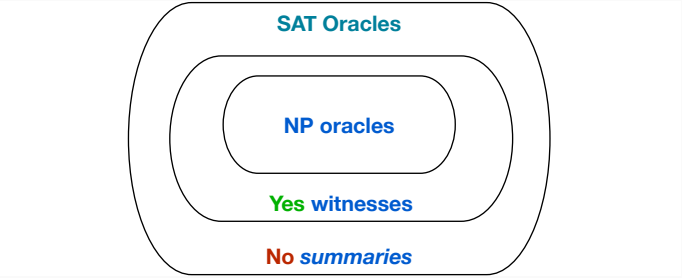
- Propositional encoding for the knapsack problem?

- **Solution:** consider 0-1 ILP (or PBO) formulation:
  - Associate propositional variable x_i with each objet i
  - $x_i = 1$ iff object i is picked

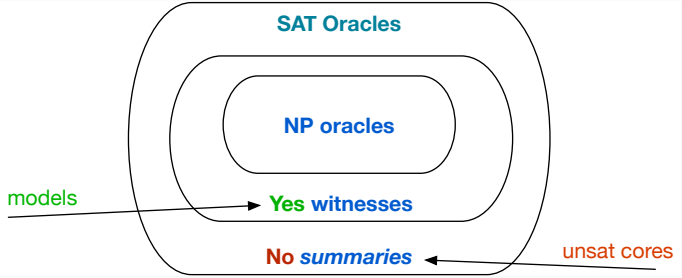
$$\begin{array}{ll} \max & \sum_{i=1}^{n} v_i \cdot x_i \\ \text{s.t} & \sum_{i=1}^{n} w_i \cdot x_i \leq W \end{array}$$

4 Problem Solving with SAT Oracles

So what are SAT oracles?



So what are SAT oracles?



Computing a model

- **Q:** How to solve the FSAT problem?
  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

Computing a model

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.

Computing a model

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\mathrm{var}(\mathcal{F})|$ calls to an NP oracle

Computing a model

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \wedge (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\mathrm{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $\mathrm{P} = \mathrm{NP}$ [GF93]

- FSAT is an example of a function problem

Computing a model

- **Q:** How to solve the FSAT problem?

  FSAT: Compute a model of a satisfiable CNF formula $\mathcal{F}$, using an NP oracle

  - A possible algorithm:
    1. Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \mathrm{var}(\mathcal{F})$, in order
    2. $i \leftarrow 1$ and $\mathcal{F}_i \triangleq \mathcal{F}$
    3. Call NP oracle on $\mathcal{F}_i \land (x_i)$
    4. If answer is **yes**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (x_i)$
    5. If answer is **no**, then $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i \cup (\neg x_i)$
    6. $i \leftarrow i + 1$
    7. If $i \leq n$, then repeat from 3.
  - Algorithm needs $|\mathrm{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $\mathrm{P} = \mathrm{NP}$ [GF93]

- FSAT is an example of a function problem
  - **Note:** FSAT can be solved with **one** SAT oracle call

Beyond decision problems

Answer	Problem Type

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	

Beyond decision problems

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

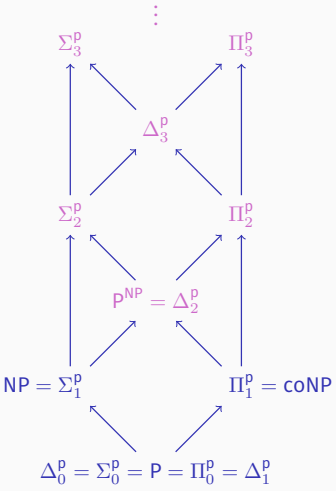
Beyond decision problems

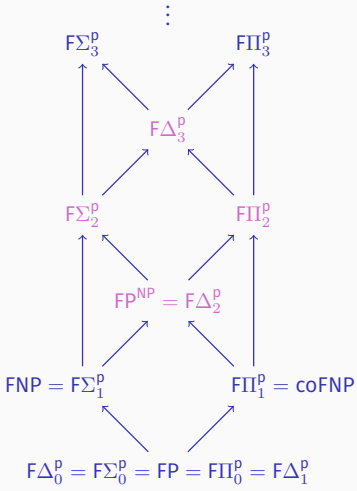
Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems
# solutions	

Beyond decision problems

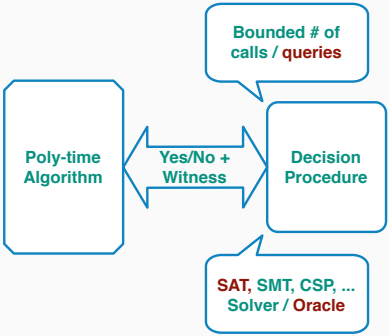
Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems
# solutions	Counting Problems

… and beyond NP – decision and function problems

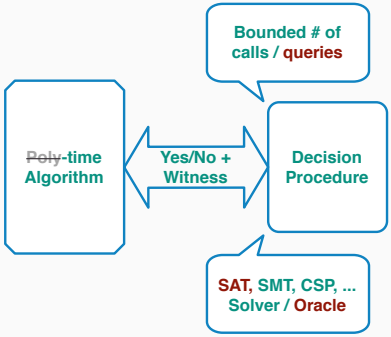




Oracle-based problem solving – simple scenario



Oracle-based problem solving – general setting

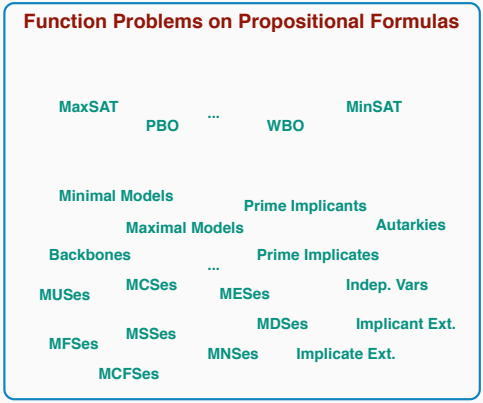


Many problems to solve – within $\mathrm{FP}^{\mathrm{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

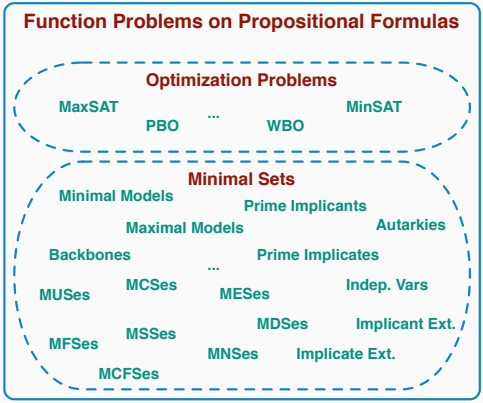
Many problems to solve – within $\mathrm{FP}^{\mathrm{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems

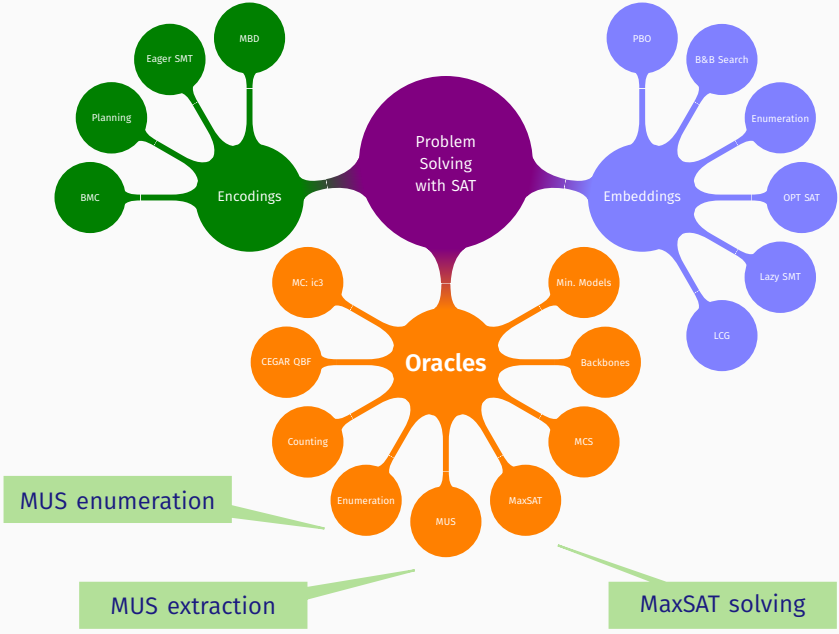


Many problems to solve – within $\mathrm{FP}^{\mathrm{NP}}$

Answer	Problem Type
Yes/No	Decision Problems
Some solution	Function Problems
All solutions	Enumeration Problems



Selection of topics



Minimal Unsatisfiability

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?
- How to compute these **minimal** sets?

Analyzing inconsistency – timetabling

Subject	Day	Time	Room
Intro Prog	Mon	9:00-10:00	6.2.46
Intro AI	Tue	10:00-11:00	8.2.37
Databases	Tue	11:00-12:00	8.2.37
... (hundreds of consistent constraints)			
Linear Alg	Mon	9:00-10:00	6.2.46
Calculus	Tue	10:00-11:00	8.2.37
Adv Calculus	Mon	9:00-10:00	8.2.06
... (hundreds of consistent constraints)			

- Set of constraints consistent / satisfiable? No
- Minimal subset of constraints that is inconsistent / unsatisfiable?
- Minimal subset of constraints whose removal makes remaining constraints consistent?
- How to compute these **minimal** sets?

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- MUSes and MCSes are (subset-)**minimal** sets
- MUSes and minimal hitting sets of MCSes and vice-versa [Rei87, BS05]
  - Easy to see **why**

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \not\models \bot$

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- Given $\mathcal{F}$ $(\models \bot)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \not\models \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \models \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ is MSS

$$(\neg x_1 \vee \neg x_2) \wedge (x_1) \wedge (x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (x_3) \wedge (x_4) \wedge (x_5 \vee x_6)$$

- MUSes and MCSes are (subset-)**minimal** sets
- MUSes and minimal hitting sets of MCSes and vice-versa [Rei87, BS05]
  - Easy to see **why**
- How to compute MUSes & MCSes **efficiently** with SAT oracles?

Why it matters?

- Analysis of over-constrained systems
  - Model-based diagnosis [Rei87]
    - Software fault localization
    - Spreadsheet debugging
    - Debugging relational specifications (e.g. Alloy)
    - Type error debugging
    - Axiom pinpointing in description logics
    - ...
  - Model checking of software & hardware systems
  - Inconsistency measurement
  - Minimal models; MinCost SAT; ...
  - ...
- Find minimal relaxations to recover consistency
  - But also minimum relaxations to recover consistency, eg. **MaxSAT**
- **Find minimal explanations of inconsistency**
  - But also minimum explanations of inconsistency, eg. Smallest MUS

Why it matters?

- Analysis of over-constrained systems
  - Model-based diagnosis [Rei87]
    - Software fault localization
    - Spreadsheet debugging
    - Debugging relational specifications (e.g. Alloy)
    - Type error debugging
    - Axiom pinpointing in description logics
    - ...
  - Model checking of software & hardware systems
  - Inconsistency measurement
  - Minimal models; MinCost SAT; ...
  - ...
- Find minimal relaxations to recover consistency
  - But also minimum relaxations to recover consistency, eg. **MaxSAT**
- **Find minimal explanations of inconsistency**
  - But also minimum explanations of inconsistency, eg. Smallest MUS

Deletion-based algorithm

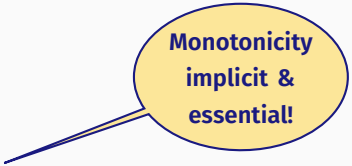
Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \mathcal{F}$
  foreach $c \in \mathcal{M}$ **do**
    if $\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // If $\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$, then $c \notin$ MUS
  return $\mathcal{M}$ // Final $\mathcal{M}$ is MUS
end

- Number of oracles calls: $\mathcal{O}(m)$

[CD91, BDTW93]

Deletion-based algorithm

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \mathcal{F}$
  foreach $c \in \mathcal{M}$ **do**
    if $\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // Remove c from $\mathcal{M}$
  return $\mathcal{M}$ // Final $\mathcal{M}$ is MUS
end



- Number of oracles calls: $\mathcal{O}(m)$

[CD91, BDTW93]

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$	Outcome

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4c_6c_7$	0	Keep c_5

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4c_6c_7$	0	Keep c_5
$c_4..c_7$	$c_4c_5c_7$	0	Keep c_6

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4c_6c_7$	0	Keep c_5
$c_4..c_7$	$c_4c_5c_7$	0	Keep c_6
$c_4..c_7$	$c_4..c_6$	1	Drop c_7

Deletion – MUS example

c_1	c_2	c_3	c_4	c_5	c_6	c_7
$(\neg x_1 \vee \neg x_2)$	(x_1)	(x_2)	$(\neg x_3 \vee \neg x_4)$	(x_3)	(x_4)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\mathrm{SAT}(\mathcal{M} \setminus \{c\})$	Outcome
$c_1..c_7$	$c_2..c_7$	1	Drop c_1
$c_2..c_7$	$c_3..c_7$	1	Drop c_2
$c_3..c_7$	$c_4..c_7$	1	Drop c_3
$c_4..c_7$	$c_5..c_7$	0	Keep c_4
$c_4..c_7$	$c_4c_6c_7$	0	Keep c_5
$c_4..c_7$	$c_4c_5c_7$	0	Keep c_6
$c_4..c_7$	$c_4..c_6$	1	Drop c_7

- MUS: $\{c_4, c_5, c_6\}$

Many MUS algorithms

- Formula $\mathcal{F}$ with m clauses k the size of largest minimal subset

Algorithm	Oracle Calls	Reference
Insertion-based	$\mathcal{O}(k\,m)$	[dSNP88, vMW08]
MCS_MUS	$\mathcal{O}(k\,m)$	[BK15]
Deletion-based	$\mathcal{O}(m)$	[CD91, BDTW93]
Linear insertion	$\mathcal{O}(m)$	[MSL11, BLM12]
Dichotomic	$\mathcal{O}(k\,\log(m))$	[HLSB06]
QuickXplain	$\mathcal{O}(k + k\,\log(\frac{m}{k}))$	[Jun04]
Progression	$\mathcal{O}(k\,\log(1+\frac{m}{k}))$	[MJB13]

- **Note:** Lower bound in $\mathrm{FP}^{\mathrm{NP}}_{||}$ and upper bound in $\mathrm{FP}^{\mathrm{NP}}$ [CT95]
- Oracle calls correspond to testing unsatisfiability with SAT solver
- Practical optimizations: clause set trimming; clause set refinement; redundancy removal; (recursive) model rotation

MUS Enumeration

How to enumerate MUSes?

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

2. Exploit recent advances in 2QBF solving

How to enumerate MUSes?

1. Standard solution:
   Exploit HS duality between MCSes and MUSes [Rei87, LS08]

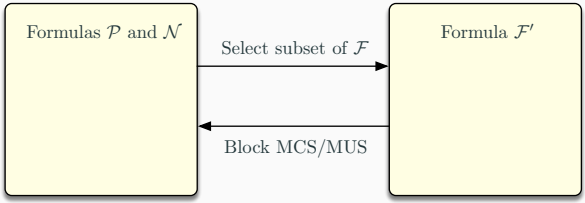
   MCSes are MHSes of MUSes and vice-versa

   - Enumerate **all** MCSes and then enumerate **all** MHSes of the MCSes, i.e. compute all the MUSes
   - Problematic if too many MCSes, and we want the MUSes
   - And, often we want to enumerate the MUSes

2. Exploit recent advances in 2QBF solving

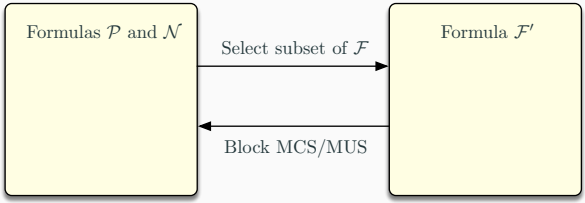
3. Implicit hitting set dualization [LPMM16]
   - Most effective if MUSes provided to user on-demand

How to enumerate MUSes, preferably?



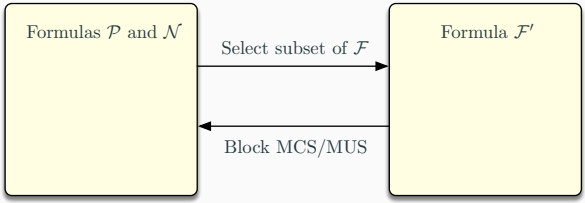
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)

How to enumerate MUSes, preferably?



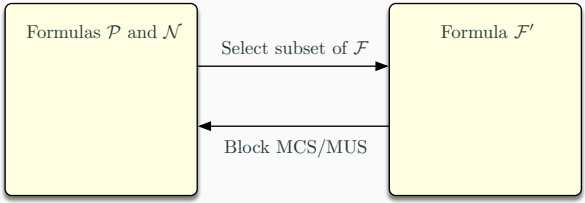
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate

How to enumerate MUSes, preferably?



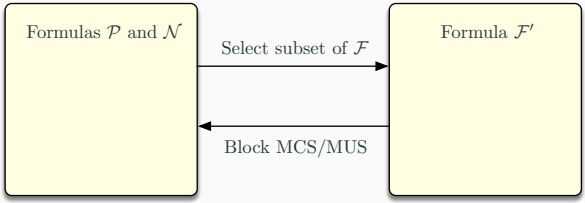
1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H

How to enumerate MUSes, preferably?



1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H
4. Run SAT oracle on $\mathcal{F}'$
   - If $\mathcal{F}'$ unsatisfiable: **extract new MUS**
   - Otherwise, H **is** already an **MCS of** $\mathcal{F}$

How to enumerate MUSes, preferably?



1. Keep sets representing computed MUSes (set $\mathcal{N}$) and MCSes (set $\mathcal{P}$)
2. Compute minimal hitting set (MHS) H of $\mathcal{N}$, subject to $\mathcal{P}$
   - Must **not** repeat MUSes
   - Must **not** repeat MCSes
   - Maximize clauses picked, i.e. prefer to check satisfiability on as **many** clauses as possible
   - If unsatisfiable: no more MUSes/MCSes to enumerate
3. Target set: $\mathcal{F}'$, i.e. $\mathcal{F}$ minus clauses from H
4. Run SAT oracle on $\mathcal{F}'$
   - If $\mathcal{F}'$ unsatisfiable: **extract new MUS**
   - Otherwise, H **is** already an **MCS of** $\mathcal{F}$
5. Repeat loop

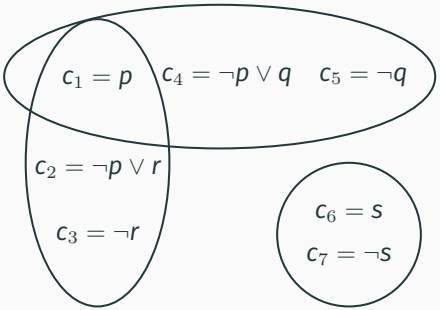
MARCO/eMUS algorithm

Input: CNF formula $\mathcal{F}$

```
1  begin
2      I ← {p_i | c_i ∈ F}
3      (P, N) ← (∅, ∅)
4      while true do
5          (st, H) ← MinHittingSet(N, P)
6          if not st then return
7          F' ← {c_i | p_i ∈ I ∧ p_i ∉ H}
8          if not SAT(F') then
9              M ← ComputeMUS(F')
10             ReportMUS (M)
11             N ← N ∪ {¬p_i | c_i ∈ M}
12         else
13             P ← P ∪ {p_i | p_i ∈ H}
14 end
```

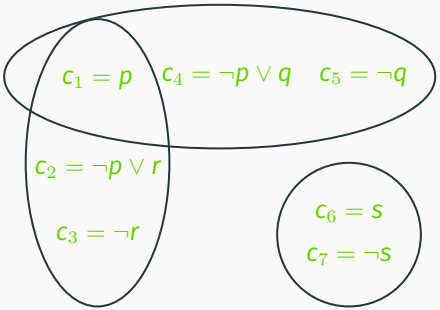
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



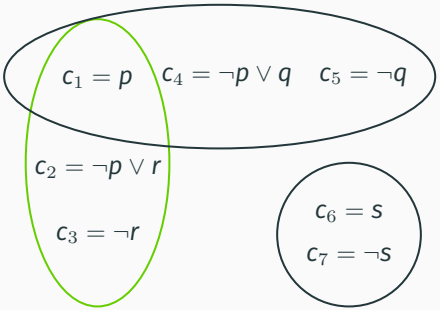
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



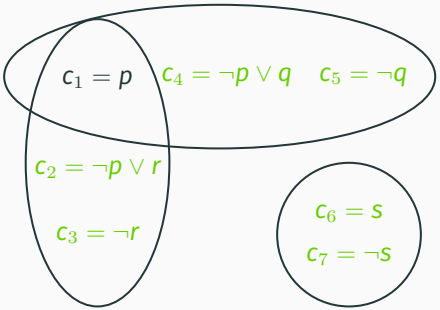
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



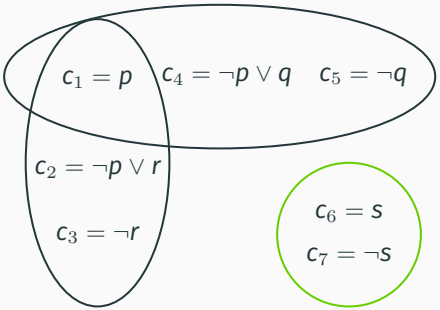
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



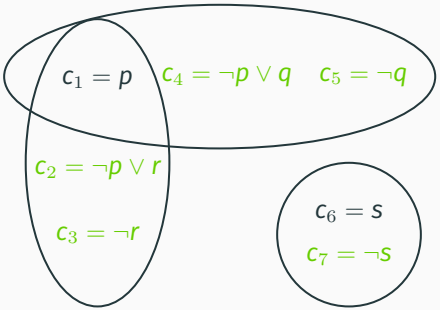
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



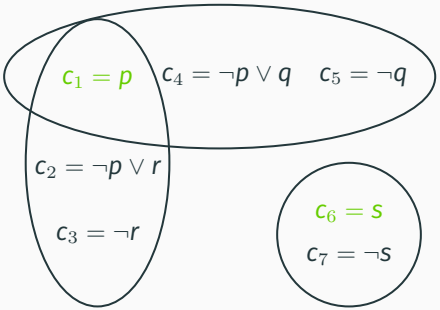
An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$



An example

MinHS ($\mathcal{N}$) $p_1p_2p_3p_4p_5p_6p_7$	$\mathcal{F}'$ S/U	MUS/MCS
1111111	U	$\neg p_1 \vee \neg p_2 \vee \neg p_3$
0111111	U	$\neg p_6 \vee \neg p_7$
0111101	S	$p_1 \vee p_6$
1011101	U	$\neg p_1 \vee \neg p_4 \vee \neg p_5$
1101010	S	$p_3 \vee p_5 \vee p_7$
1010110	S	$p_2 \vee p_4 \vee p_7$
1100101	S	$p_3 \vee p_4 \vee p_6$
0111110	S	$p_1 \vee p_7$
1101001	S	$p_3 \vee p_5 \vee p_6$
1010101	S	$p_2 \vee p_4 \vee p_6$
1011001	S	$p_2 \vee p_5 \vee p_6$
1100110	S	$p_3 \vee p_4 \vee p_7$
1011010	S	$p_2 \vee p_5 \vee p_7$

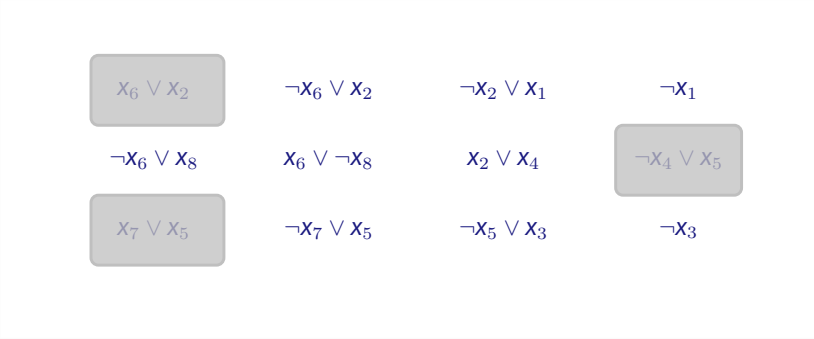


Maximum Satisfiability

Recap MaxSAT

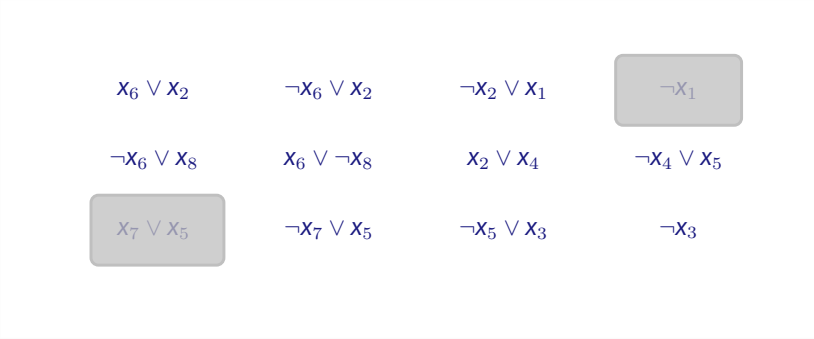
- Given unsatisfiable formula, find largest subset of clauses that is satisfiable

Recap MaxSAT



- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula

Recap MaxSAT



- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes

Recap MaxSAT

$$x_6 \vee x_2 \qquad \neg x_6 \vee x_2 \qquad \neg x_2 \vee x_1 \qquad \neg x_1$$

$$\neg x_6 \vee x_8 \qquad x_6 \vee \neg x_8 \qquad x_2 \vee x_4 \qquad \neg x_4 \vee x_5$$

$$x_7 \vee x_5 \qquad \neg x_7 \vee x_5 \qquad \neg x_5 \vee x_3 \qquad \neg x_3$$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest MCSes
  - **Note**: Clauses can have weights & there can be hard clauses

Recap MaxSAT

$$x_6 \vee x_2 \qquad \neg x_6 \vee x_2 \qquad \neg x_2 \vee x_1 \qquad \neg x_1$$

$$\neg x_6 \vee x_8 \qquad x_6 \vee \neg x_8 \qquad x_2 \vee x_4 \qquad \neg x_4 \vee x_5$$

$$x_7 \vee x_5 \qquad \neg x_7 \vee x_5 \qquad \neg x_5 \vee x_3 \qquad \neg x_3$$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest **cost** MCSes
  - **Note**: Clauses can have weights & there can be hard clauses

Recap MaxSAT

$$x_6 \vee x_2 \qquad \neg x_6 \vee x_2 \qquad \neg x_2 \vee x_1 \qquad \neg x_1$$

$$\neg x_6 \vee x_8 \qquad x_6 \vee \neg x_8 \qquad x_2 \vee x_4 \qquad \neg x_4 \vee x_5$$

$$x_7 \vee x_5 \qquad \neg x_7 \vee x_5 \qquad \neg x_5 \vee x_3 \qquad \neg x_3$$

- Given unsatisfiable formula, find largest subset of clauses that is satisfiable
- A Minimal Correction Subset (**MCS**) is an irreducible relaxation of the formula
- The MaxSAT solution is one of the smallest **cost** MCSes
  - **Note**: Clauses can have weights & there can be hard clauses
- **Many** practical applications

[SZGN17]

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No		
	Yes		

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

- Must satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost
  (s.t. hard & remaining soft clauses are satisfied)

MaxSAT problem(s)

		Hard Clauses?	
		No	Yes
Weights?	No	Plain	Partial
	Yes	Weighted	Weighted Partial

- Must satisfy hard clauses, if any
- Compute set of satisfied soft clauses with maximum cost
  - Without weights, cost of each falsified soft clause is 1
- **Or**, compute set of falsified soft clauses with minimum cost
  (s.t. hard & remaining soft clauses are satisfied)
- **Note**: goal is to compute **set** of satisfied (or falsified) clauses;
  not just the cost **!**

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - Is 2 the MaxSAT solution??

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - Is 2 the MaxSAT solution??
  - **No!** Enough to either falsify (x) or (z)

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - Is 2 the MaxSAT solution??
  - **No!** Enough to either falsify (x) or (z)
- Cannot use unit propagation

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - Is 2 the MaxSAT solution??
  - **No!** Enough to either falsify (x) or (z)
- Cannot use unit propagation
- Cannot learn clauses (using unit propagation)

Issues with MaxSAT

- **Unit propagation is unsound for MaxSAT**
  - Formula with all clauses soft:

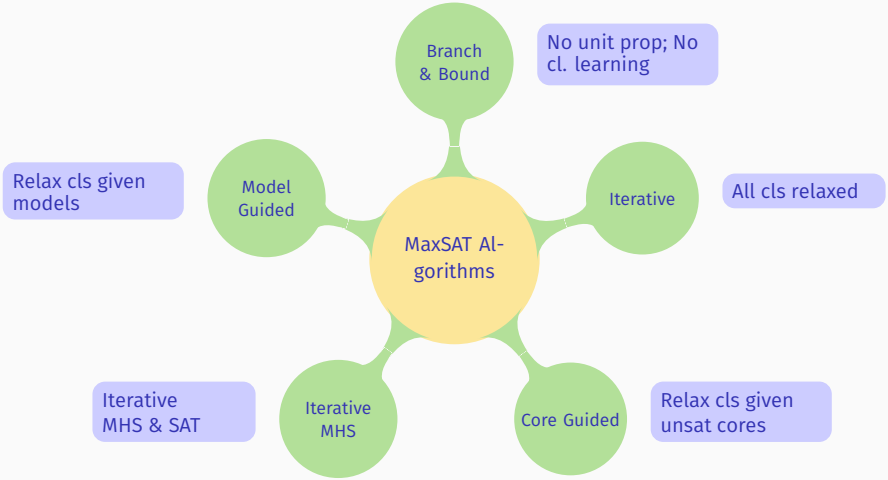
$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

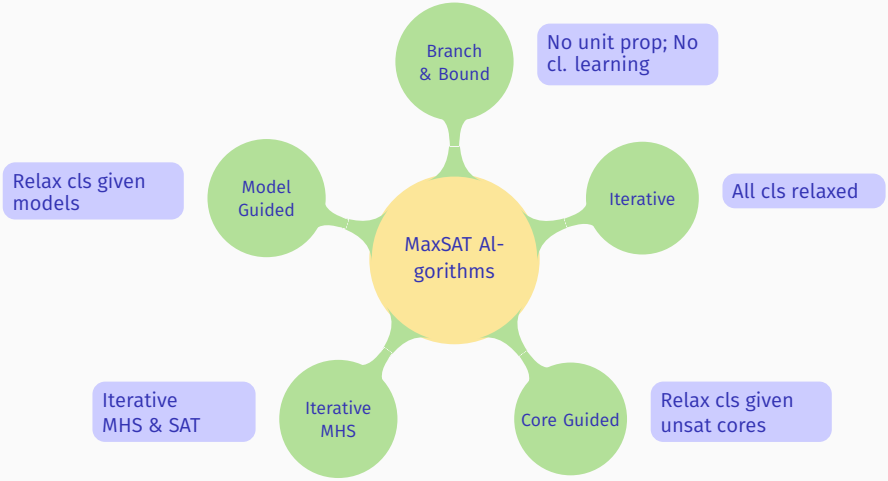
$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - Is 2 the MaxSAT solution??
  - **No!** Enough to either falsify (x) or (z)
- Cannot use unit propagation
- Cannot learn clauses (using unit propagation)
- Need to solve MaxSAT using different techniques

Many MaxSAT approaches



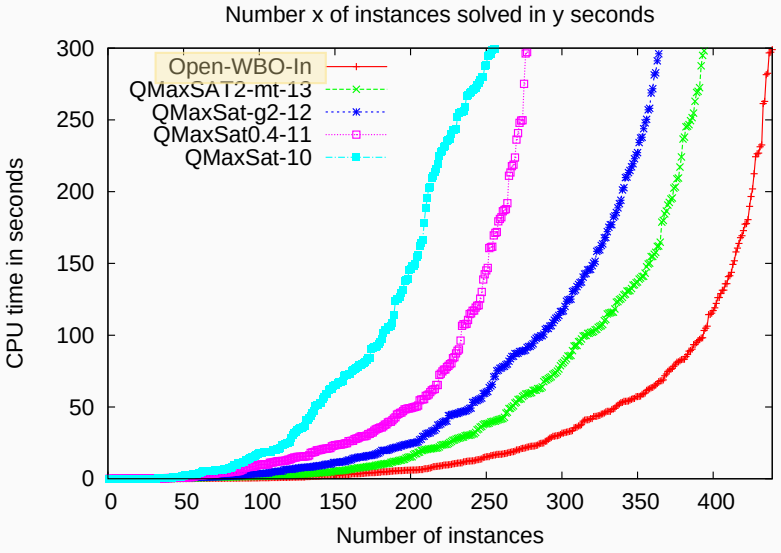
Many MaxSAT approaches



- For practical (**industrial**) instances: **core-guided** & **iterative MHS** approaches are the most effective

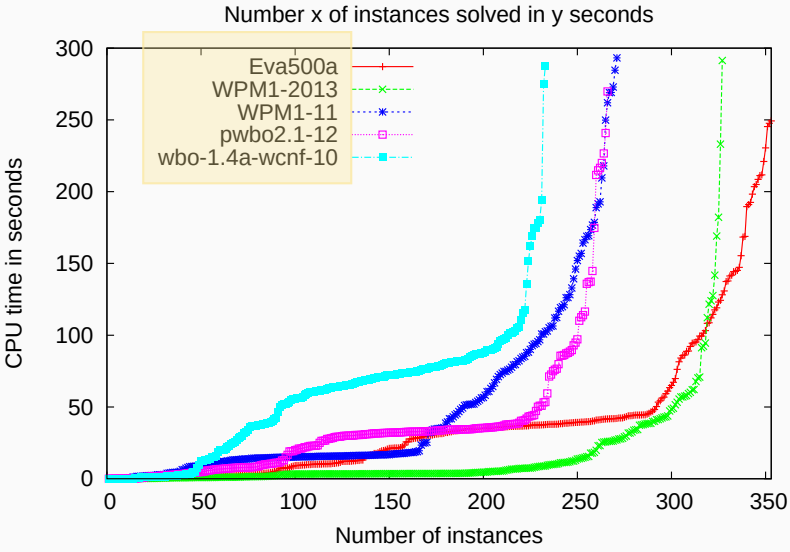
[MaxSAT14]

Core-guided solver performance – partial



Source: [MaxSAT 2014 organizers]

Core-guided solver performance – weighted partial



Source: [MaxSAT 2014 organizers]

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \qquad \neg x_6 \vee x_2 \qquad \neg \neg x_2 \vee x_1 \qquad \neg x_1$$

$$\neg x_6 \vee x_8 \qquad x_6 \vee \neg x_8 \qquad x_2 \vee x_4 \qquad \neg x_4 \vee x_5$$

$$x_7 \vee x_5 \qquad \neg x_7 \vee x_5 \qquad \neg x_5 \vee x_3 \qquad \neg x_3$$

Example CNF formula

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 12$$

Relax all clauses; Set $UB = 12 + 1$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\quad \neg x_6 \vee x_2 \vee r_2$ $\quad \neg x_2 \vee x_1 \vee r_3$ $\quad \neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\quad x_6 \vee \neg x_8 \vee r_6$ $\quad x_2 \vee x_4 \vee r_7$ $\quad \neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_{11}$ $\quad \neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 12$

Formula is SAT; E.g. all $x_i = 0$ and $r_1 = r_7 = r_9 = 1$ (i.e. cost $= 3$)

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 2$$

Refine $UB = 3$

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 2$$

Formula is SAT; E.g. $x_1 = x_2 = 1$; $x_3 = \ldots = x_8 = 0$ and $r_4 = r_9 = 1$ (i.e. cost $= 2$)

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 1$$

Refine $UB = 2$

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 1$$

Formula is UNSAT; terminate

Basic MaxSAT with iterative SAT solving

$$x_6 \vee x_2 \vee r_1 \qquad \neg x_6 \vee x_2 \vee r_2 \qquad \neg x_2 \vee x_1 \vee r_3 \qquad \neg x_1 \vee r_4$$

$$\neg x_6 \vee x_8 \vee r_5 \qquad x_6 \vee \neg x_8 \vee r_6 \qquad x_2 \vee x_4 \vee r_7 \qquad \neg x_4 \vee x_5 \vee r_8$$

$$x_7 \vee x_5 \vee r_9 \qquad \neg x_7 \vee x_5 \vee r_{10} \qquad \neg x_5 \vee x_3 \vee r_{11} \qquad \neg x_3 \vee r_{12}$$

$$\sum_{i=1}^{12} r_i \leq 1$$

MaxSAT solution is last satisfied UB: $UB = 2$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$ $\quad \neg x_6 \vee x_2 \vee r_2$ $\quad \neg x_2 \vee x_1 \vee r_3$ $\quad \neg x_1 \vee r_4$

$\neg x_6 \vee x_8 \vee r_5$ $\quad x_6 \vee \neg x_8 \vee r_6$ $\quad x_2 \vee x_4 \vee r_7$ $\quad \neg x_4 \vee x_5 \vee r_8$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_{11}$ $\quad \neg x_3 \vee r_{12}$

$\sum_{i=1}^{12} r_i \leq 1$

MaxSAT solution is last satisfied UB: $UB = 2$

AtMost$k$/PB constraints over **all** relaxation variables

All (possibly many) soft clauses relaxed

MSU3 core-guided algorithm

$$x_6 \vee x_2 \qquad \neg x_6 \vee x_2 \qquad \neg x_2 \vee x_1 \qquad \neg x_1$$

$$\neg x_6 \vee x_8 \qquad x_6 \vee \neg x_8 \qquad x_2 \vee x_4 \qquad \neg x_4 \vee x_5$$

$$x_7 \vee x_5 \qquad \neg x_7 \vee x_5 \qquad \neg x_5 \vee x_3 \qquad \neg x_3$$

Example CNF formula

MSU3 core-guided algorithm

$x_6 \vee x_2$ $\quad$ $\neg x_6 \vee x_2$ $\quad$ $\neg x_2 \vee x_1$ $\quad$ $\neg x_1$

$\neg x_6 \vee x_8$ $\quad$ $x_6 \vee \neg x_8$ $\quad$ $x_2 \vee x_4$ $\quad$ $\neg x_4 \vee x_5$

$x_7 \vee x_5$ $\quad$ $\neg x_7 \vee x_5$ $\quad$ $\neg x_5 \vee x_3$ $\quad$ $\neg x_3$

Formula is UNSAT; OPT $\leq |\varphi| - 1$; Get unsat core

MSU3 core-guided algorithm

$x_6 \vee x_2$ $\quad \neg x_6 \vee x_2$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

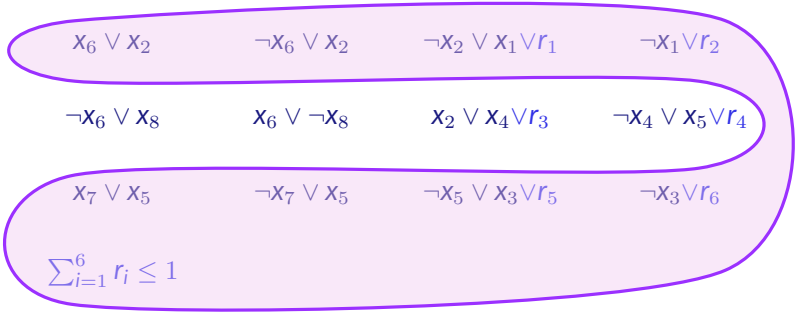
$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5$ $\quad \neg x_7 \vee x_5$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{6} r_i \leq 1$

Add relaxation variables and AtMost$k$, $k = 1$, constraint

MSU3 core-guided algorithm



Formula is (again) UNSAT; OPT $\leq |\varphi| - 2$; Get unsat core

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

Add new relaxation variables and update AtMost*k*, k=2, constraint

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

Instance is now SAT

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\varphi| - \mathcal{I} = 12 - 2 = 10$

AtMost$k$/PB constraints used

Relaxed soft clauses become **hard**

MSU3 core-guided algorithm

$x_6 \vee x_2 \vee r_7$ $\quad \neg x_6 \vee x_2 \vee r_8$ $\quad \neg x_2 \vee x_1 \vee r_1$ $\quad \neg x_1 \vee r_2$

$\neg x_6 \vee x_8$ $\quad x_6 \vee \neg x_8$ $\quad x_2 \vee x_4 \vee r_3$ $\quad \neg x_4 \vee x_5 \vee r_4$

$x_7 \vee x_5 \vee r_9$ $\quad \neg x_7 \vee x_5 \vee r_{10}$ $\quad \neg x_5 \vee x_3 \vee r_5$ $\quad \neg x_3 \vee r_6$

$\sum_{i=1}^{10} r_i \leq 2$

MaxSAT solution is $|\rho| - \mathcal{I} = 12 - 2 = 10$

AtMost$k$/PB constraints used

Some clauses not relaxed

Relaxed soft clauses become **hard**

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)?

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \emptyset$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No
- Core of $\mathcal{F}$: $\{c_1, c_2, c_3, c_4\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: $\emptyset$
- SAT($\mathcal{F} \setminus \emptyset$)? No
- Core of $\mathcal{F}$: $\{c_1, c_2, c_3, c_4\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)?

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No
- Core of $\mathcal{F}$: $\{c_9, c_{10}, c_{11}, c_{12}\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1\}$
- SAT($\mathcal{F} \setminus \{c_1\}$)? No
- Core of $\mathcal{F}$: $\{c_9, c_{10}, c_{11}, c_{12}\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)?

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No
- Core of $\mathcal{F}$: $\{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_1, c_9\}$
- SAT($\mathcal{F} \setminus \{c_1, c_9\}$)? No
- Core of $\mathcal{F}$: $\{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}$. Update $\mathcal{K}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$:

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)?

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)? Yes

MHS approach for MaxSAT

$$c_1 = x_6 \vee x_2 \qquad c_2 = \neg x_6 \vee x_2 \qquad c_3 = \neg x_2 \vee x_1 \qquad c_4 = \neg x_1$$

$$c_5 = \neg x_6 \vee x_8 \qquad c_6 = x_6 \vee \neg x_8 \qquad c_7 = x_2 \vee x_4 \qquad c_8 = \neg x_4 \vee x_5$$

$$c_9 = x_7 \vee x_5 \qquad c_{10} = \neg x_7 \vee x_5 \qquad c_{11} = \neg x_5 \vee x_3 \qquad c_{12} = \neg x_3$$

$$\mathcal{K} = \{\{c_1, c_2, c_3, c_4\}, \{c_9, c_{10}, c_{11}, c_{12}\}, \{c_3, c_4, c_7, c_8, c_{11}, c_{12}\}\}$$

- Find MHS of $\mathcal{K}$: E.g. $\{c_4, c_9\}$
- SAT($\mathcal{F} \setminus \{c_4, c_9\}$)? Yes
- Terminate & return 2

MaxSAT solving with SAT oracles – a sample

- A sample of recent algorithms:

Algorithm	# Oracle Queries	Reference
Linear search SU	Exponential***	[BP10]
Binary search	Linear*	[FM06]
FM/WMSU1/WPM1	Exponential**	[FM06, MP08, MMSP09, ABL09, ABGL12]
WPM2	Exponential**	[ABL10a, ABL13]
Bin-Core-Dis	Linear	[HMM11, MHM12]
Iterative MHS	Exponential	[DB11, DB13a, DB13b]

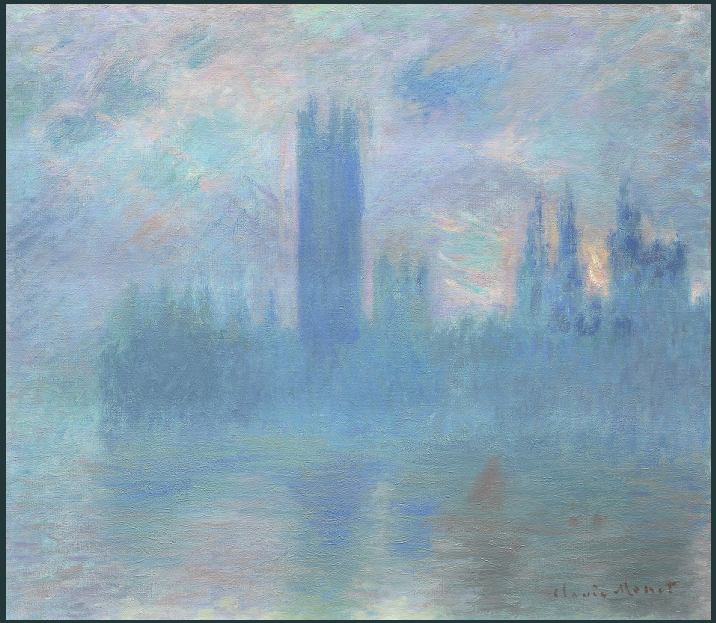
* $\mathcal{O}(\log m)$ queries with SAT oracle, for (partial) unweighted MaxSAT

** Weighted case; depends on computed cores

*** On # bits of problem instance (due to weights)

- But also additional recent work:
  - Progression [IMM+14]
  - Soft cardinality constraints (OLL) [MDM14, MIM14]
    - Recent implementation (RC2, using PySAT) won 2018 MaxSAT Evaluation
  - MaxSAT resolution [NB14]
  - ...

5 Sample of Applications



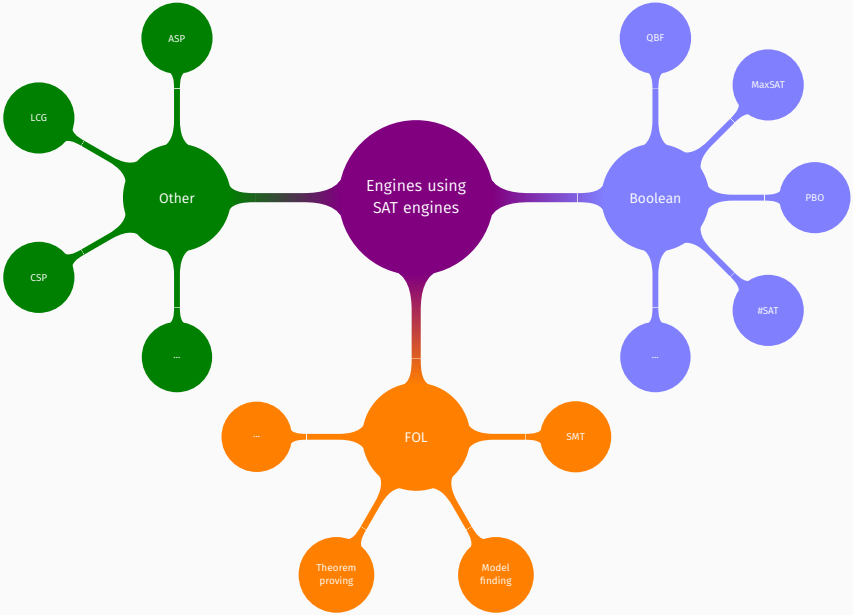
Flagship applications

- Bounded (& unbounded) model checking
- Automated planning
- Multi-agent path finding

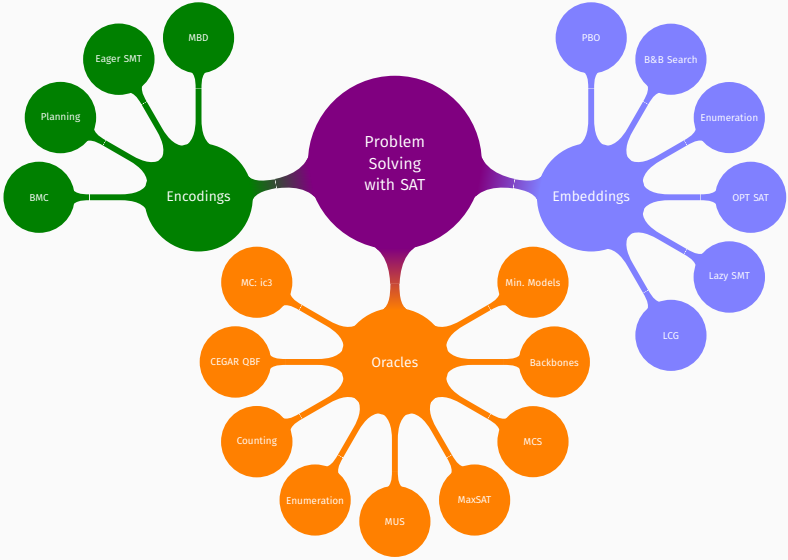
- Software model checking
- Package management
- Design debugging

- Haplotyping

CDCL SAT is the engines' engine



CDCL SAT is ubiquitous in problem solving



Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!
- **Maximum cliques with SAT** [IMM17]

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!
- **Maximum cliques with SAT** [IMM17]
- **eXplainable AI**

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!
- **Maximum cliques with SAT** [IMM17]
- **eXplainable AI**
  - **Explainable decision sets** [IPNM18]
    - Computation of smallest decision sets (rules)

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!
- **Maximum cliques with SAT** [IMM17]
- **eXplainable AI**
  - **Explainable decision sets** [IPNM18]
    - Computation of smallest decision sets (rules)
  - **Smallest (explainable) decision trees** [NIPM18]
    - Computation of smallest decision trees

Recent applications

- **Package dependency and upgradability** [IJM14]
  - Exact and approximate problem solving with SAT
- **Two-level logic minimization with SAT** [IPM15]
  - Reimplementation of Quine-McCluskey with SAT oracles
- **Model-based diagnosis** [MJIM15, IMWM19]
  - MaxSAT + *implicit hitting set dualization* using SAT oracles
  - our talk on August 14!
- **Maximum cliques with SAT** [IMM17]
- **eXplainable AI**
  - **Explainable decision sets** [IPNM18]
    - Computation of smallest decision sets (rules)
  - **Smallest (explainable) decision trees** [NIPM18]
    - Computation of smallest decision trees
  - **Abduction-based explanations for ML models** [INMS19]
    - Extraction of explanations for ***any*** ML model **on demand**

Package dependency

Package dependency

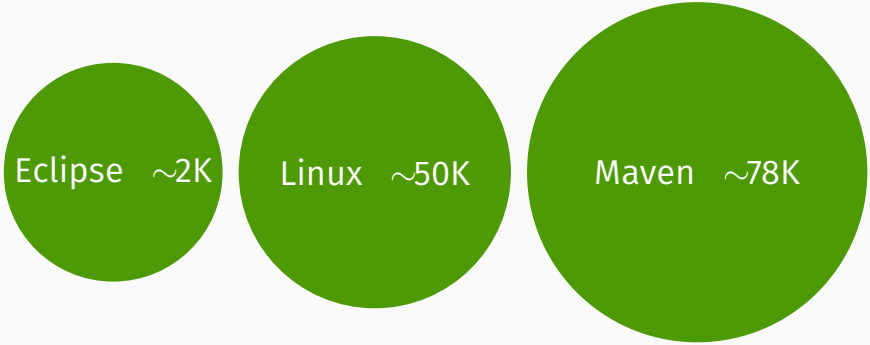


Figure 1: Number of packages in modern package management systems

Package dependency

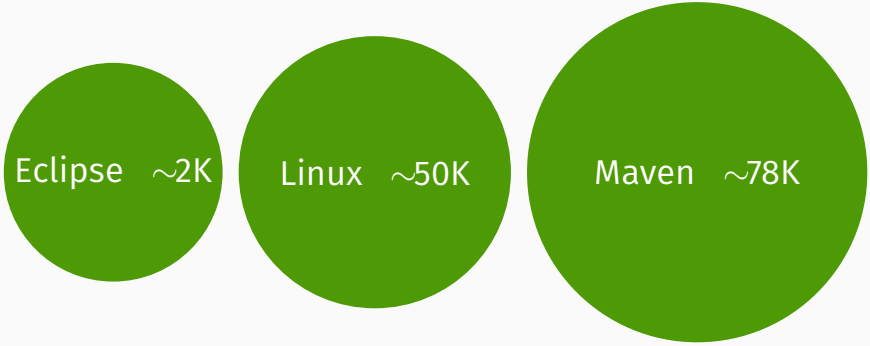
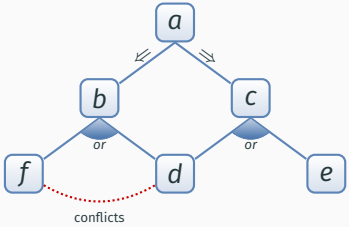


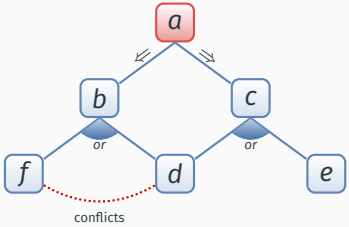
Figure 1: Number of packages in modern package management systems

can single package *P* **be installed** in repository *R*? — **NP-complete!**

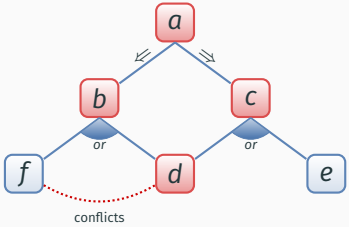
PackUP example



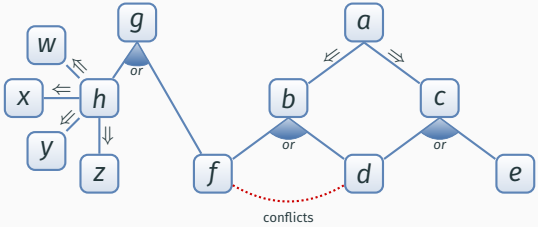
PackUP example



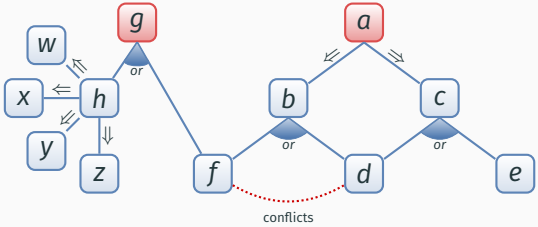
PackUP example



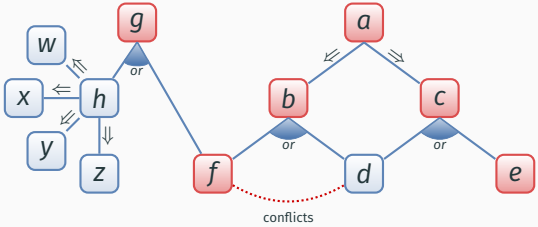
PackUP example



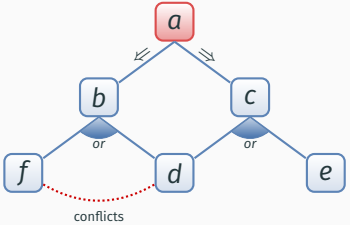
PackUP example



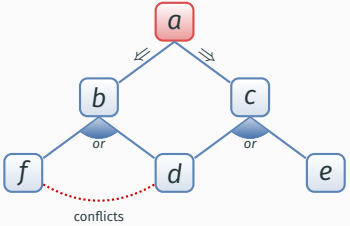
PackUP example



Encoding PackUP to SAT and MaxSAT



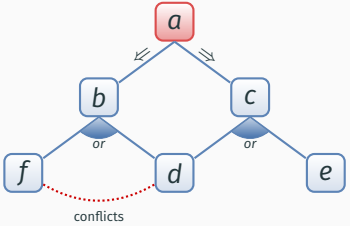
Encoding PackUP to SAT and MaxSAT



Find any solution — SAT

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee f \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg f \vee \neg d) \wedge (a)$$

Encoding PackUP to SAT and MaxSAT



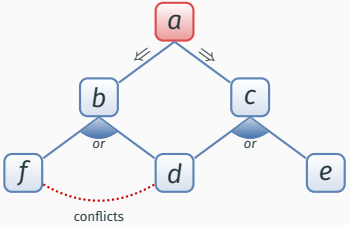
Find any solution — SAT

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee f \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg f \vee \neg d) \wedge (a)$$

Find best solution — MaxSAT

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee f \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg f \vee \neg d) \wedge (a)$$

Encoding PackUP to SAT and MaxSAT



Find any solution — SAT

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee f \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg f \vee \neg d) \wedge (a)$$

Find best solution — MaxSAT

$$(\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee f \vee d) \wedge (\neg c \vee d \vee e) \wedge (\neg f \vee \neg d) \wedge (a)$$
$$(\neg a) \wedge (\neg b) \wedge (\neg c) \wedge (\neg d) \wedge (\neg e) \wedge (\neg f)$$

Solving MaxClique with SAT

Modeling MaxClique with SAT

- given a graph $G = (V, E)$, find a **largest complete subgraph**

- **main constraint:**

  $(u, v) \notin E \quad \Rightarrow$

Modeling MaxClique with SAT

- given a graph $G = (V, E)$, find a **largest complete subgraph**

- **main constraint:**

  $(u, v) \notin E \;\Rightarrow\;$ **either** u **or** v **is not** in the maximum-size clique

Modeling MaxClique with SAT

- given a graph $G = (V, E)$, find a **largest complete subgraph**

- **main constraint:**

  $(u, v) \notin E \;\Rightarrow\;$ **either** u **or** v **is not** in the maximum-size clique

- associate Boolean x_u with $u \in V$

Modeling MaxClique with SAT

- given a graph $G = (V, E)$, find a **largest complete subgraph**
- **main constraint:**

  $(u, v) \notin E \;\Rightarrow\;$ **either** u **or** v **is not** in the maximum-size clique
- associate Boolean x_u with $u \in V$
- **main goal** — **maximize** $\sum_{u \in V} x_u$
  - e.g. use MaxSAT

An example

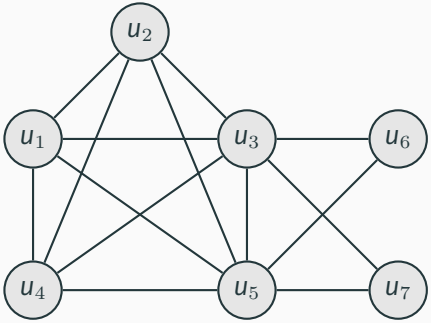
Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

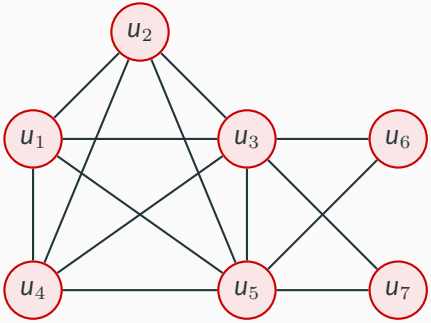
An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$



An example

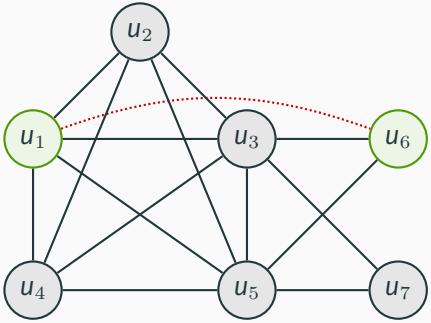
Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$



$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\,(x_2)\,(x_3) \\ (x_4)\,(x_5)\,(x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

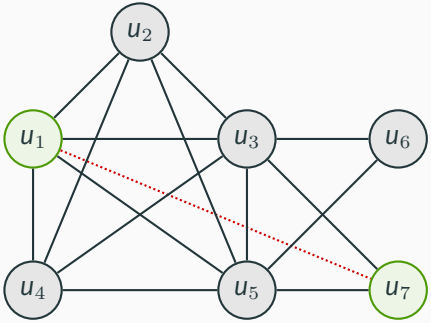


$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\,(\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\,(\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\,(\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\,(x_2)\,(x_3) \\ (x_4)\,(x_5)\,(x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

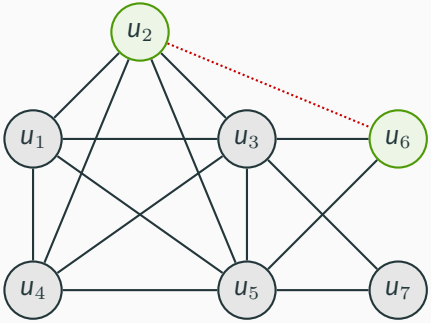


$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\ (\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\ (\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\ (\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\ (x_2)\ (x_3) \\ (x_4)\ (x_5)\ (x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

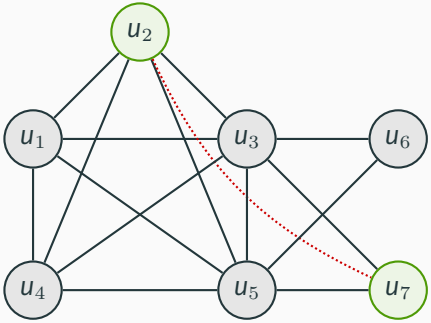


$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\,(\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\,(\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\,(\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\,(x_2)\,(x_3) \\ (x_4)\,(x_5)\,(x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

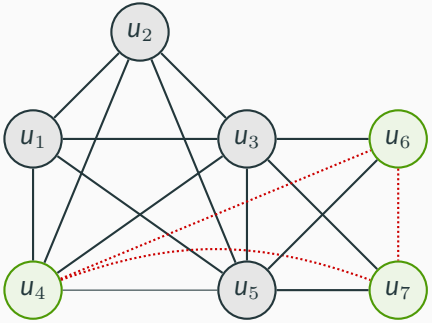


$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\,(\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\,(\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\,(\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\,(x_2)\,(x_3) \\ (x_4)\,(x_5)\,(x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u, v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$

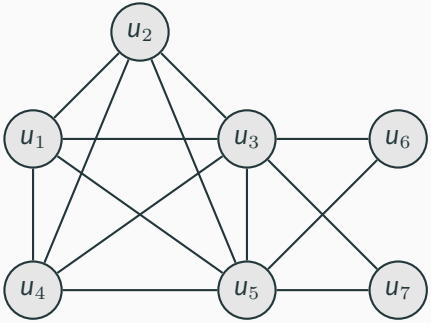


$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\,(\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\,(\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\,(\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\,(x_2)\,(x_3) \\ (x_4)\,(x_5)\,(x_6) \\ (x_7) \end{array} \right\}$$

An example

Construct $\mathcal{F} = \langle \mathcal{H}, \mathcal{S} \rangle$ s.t. $\begin{cases} \mathcal{H} & \triangleq & \{(\neg x_u \vee \neg x_v) \mid (u,v) \in E^C\} \\ \mathcal{S} & \triangleq & \{(x_u) \mid v \in V\} \end{cases}$



$$\mathcal{H} = \left\{ \begin{array}{l} (\neg x_1 \vee \neg x_6)\ (\neg x_1 \vee \neg x_7) \\ (\neg x_2 \vee \neg x_6)\ (\neg x_2 \vee \neg x_7) \\ (\neg x_4 \vee \neg x_6)\ (\neg x_4 \vee \neg x_7) \\ (\neg x_6 \vee \neg x_7) \end{array} \right\}$$

$$\mathcal{S} = \left\{ \begin{array}{l} (x_1)\ (x_2)\ (x_3) \\ (x_4)\ (x_5)\ (x_6) \\ (x_7) \end{array} \right\}$$

solve $\mathcal{F}$ with MaxSAT !

But the size of E^C can be problematic...

Instance	$\|V\|$	$\|E\|$	$\|E\|^C$
comm-n10000	10000	10000	49995000
ca-AstroPh	18772	396160	175807218
ca-citeseer	227322	814136	25836945367
ca-coauthors-dblp	540488	15245731	146048663585
ca-CondMat	23133	186936	267392475
ca-dblp-2010	226415	716462	25631272858
ca-dblp-2012	317082	1049868	50269606035
ca-HepPh	12008	237010	71865026
ca-HepTh	9877	51971	48730532
ca-MathSciNet	332689	820644	55340331061
ia-email-EU	32430	54397	525814268
ia-reality-call	6809	9484	23175161
ia-retweet-pol	18470	61157	170518528
ia-wiki-Talk	92117	360767	4242456136
rt-pol	18470	61157	170518528
rt_barackobama	9631	9826	46373070
soc-epinions	63947	606512	2044034866
soc-gplus	23628	39242	279113764
tech-as-caida2007	26477	53383	350475620
tech-internet-as	40164	85123	806508407
tech-pgp	10680	24340	57012200
tech-WHOIS	7476	56943	27892083
web-arabic-2005	163598	1747269	13380487332
web-baidu-baike-related	415641	3284387	86375643874
web-it-2004	509338	7178413	129705675378
web-NotreDame	325729	1497134	53048356451
web-sk-2005	121422	334419	7371377334

But the size of E^C can be problematic...

Instance	$\|V\|$	$\|E\|$	$\|E\|^C$
comm-n10000	10000	10000	49995000
ca-AstroPh	18772	396160	175807218
ca-citeseer	227322	814136	25836945367
ca-coauthors-dblp	540488	15245731	146048663585
ca-CondMat	23133	186936	267392475
ca-dblp-2010	226415	716462	25631272858
ca-dblp-2012	317082	1049868	50269606035
ca-HepPh	12008	237010	71865026
ca-HepTh	9877	51971	48730532
ca-MathSciNet	332689	820644	55340331061
ia-email-EU	32430	54397	525814268
ia-reality-call	6809	9484	23175161
ia-retweet-pol	18470	61157	170518528
ia-wiki-Talk	92117	360767	4242456136
rt-pol	18470	61157	170518528
rt_barackobama	9631	9826	46373070
soc-epinions	63947	606512	2044034866
soc-gplus	23628	39242	279113764
tech-as-caida2007	26477	53383	350475620
tech-internet-as	40164	85123	806508407
tech-pgp	10680	24340	57012200
tech-WHOIS	7476	56943	27892083
web-arabic-2005	163598	1747269	13380487332
web-baidu-baike-related	415641	3284387	86375643874
web-it-2004	509338	7178413	129705675378
web-NotreDame	325729	1497134	53048356451
web-sk-2005	121422	334419	7371377334

$$|E^C| = \frac{|E| \times (|E|-1)}{2} - |E|$$

But the size of E^C can be problematic...

Instance	$\|V\|$	$\|E\|$	$\|E\|^C$
comm-n10000	10000	10000	49995000
ca-AstroPh	18772	396160	175807218
ca-citeseer	227322	814136	25836945367
ca-coauthors-dblp	540488	15245731	146048663585
ca-CondMat	23133	186936	267392475
ca-dblp-2010	226415	716462	25631272858
ca-dblp-2012	317082	1049868	50269606035
ca-HepPh	12008	237010	71865026
ca-HepTh	9877	51971	48730532
ca-MathSciNet	332689	820644	55340331061
ia-email-EU	32430	54397	525814268
ia-reality-call	6809	9484	23175161
ia-retweet-pol	18470	61157	170518528
ia-wiki-Talk	92117	360767	4242456136
rt-pol	18470	61157	170518528
rt_barackobama	9631	9826	46373070
soc-epinions	63947	606512	2044034866
soc-gplus	23628	39242	279113764
tech-as-caida2007	26477	53383	350475620
tech-internet-as	40164	85123	806508407
tech-pgp	10680	24340	57012200
tech-WHOIS	7476	56943	27892083
web-arabic-2005	163598	1747269	13380487332
web-baidu-baike-related	415641	3284387	86375643874
web-it-2004	509338	7178413	129705675378
web-NotreDame	325729	1497134	53048356451
web-sk-2005	121422	334419	7371377334

$$|E^C| = \frac{|E| \times (|E|-1)}{2} - |E|$$

Unrealistic to model with SAT on sparse graphs

How to reduce the encoding size?

- **main hurdle:**
  - approaches based on $G^C = (V, E^C)$ **will not** scale...
  - and $G = (V, E)$ is **much smaller** than $G^C = (V, E^C)$

How to reduce the encoding size?

- **main hurdle:**
  - approaches based on $G^C = (V, E^C)$ **will not** scale...
  - and $G = (V, E)$ is **much smaller** than $G^C = (V, E^C)$

- can we model MaxClique using **solely** G?

Another take at solving MaxClique with SAT

revisit the original **decision problem**:

given $G = (V, E)$, is there a clique of **size** K?

Another take at solving MaxClique with SAT

revisit the original **decision problem**:
given $G = (V, E)$, is there a clique of **size** K?

1. one **must** pick exactly K vertices:

$$\sum_{u \in V} x_u = K$$

Another take at solving MaxClique with SAT

revisit the original **decision problem**:
given $G = (V, E)$, is there a clique of **size** K?

1. one **must** pick exactly K vertices:

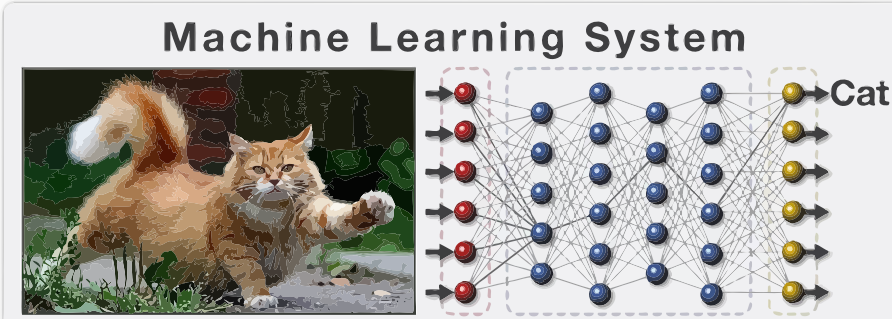
$$\sum_{u \in V} x_u = K$$

2. if a vertex $u \in V$ is picked (i.e. $x_u = 1$), **then** $K - 1$ of its neighbours **must** also be picked:

$$x_u \rightarrow \left(\sum_{v \in \mathrm{Adj}(u)} x_v = K - 1 \right)$$

eXplainable AI (XAI)

What is eXplainable AI (XAI)?



interpretable ML models

(decision trees, lists, sets)

interpretable ML models
(decision trees, lists, sets)

explanation of ML models **“on the fly”**

Interpretable ML models

Example	Lecture	Concert	Expo	Shop	*Hike?*
e_1	1	0	1	0	0
e_2	1	0	0	1	0
e_3	0	0	1	0	1
e_4	1	1	0	0	0
e_5	0	0	0	1	1
e_6	1	1	1	1	0
e_7	0	1	1	0	0
e_8	0	0	1	1	1

(a) When should we **hike** and when **not**?

Interpretable ML models

Example	Lecture	Concert	Expo	Shop	*Hike?*
e_1	1	0	1	0	0
e_2	1	0	0	1	0
e_3	0	0	1	0	1
e_4	1	1	0	0	0
e_5	0	0	0	1	1
e_6	1	1	1	1	0
e_7	0	1	1	0	0
e_8	0	0	1	1	1

(a) When should we **hike** and when **not**?

if ¬**Lecture** **and** ¬**Concert** **then** *Hike*
if **Lecture** **then** ¬*Hike*
if **Concert** **then** ¬*Hike*

(b) Example decision set

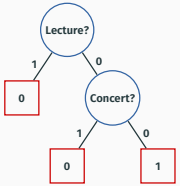
Interpretable ML models

Example	Lecture	Concert	Expo	Shop	*Hike?*
e_1	1	0	1	0	0
e_2	1	0	0	1	0
e_3	0	0	1	0	1
e_4	1	1	0	0	0
e_5	0	0	0	1	1
e_6	1	1	1	1	0
e_7	0	1	1	0	0
e_8	0	0	1	1	1

(a) When should we **hike** and when **not**?

if ¬**Lecture** **and** ¬**Concert** **then** *Hike*
if **Lecture** **then** ¬*Hike*
if **Concert** **then** ¬*Hike*

(b) Example decision set



(c) Example decision tree

[IPNM18, NIPM18]

can be **encoded** into SAT

a series of SAT oracle calls

Smallest decision trees – encoding sizes in bytes

[NIPM18]

Model	Weather	Mouse	Cancer	Car	Income
CP'09*	27K	3.5M	92G	842M	354G

Smallest decision trees – encoding sizes in bytes

[NIPM18]

Model	Weather	Mouse	Cancer	Car	Income
CP'09*	27K	3.5M	92G	842M	354G
IJCAI'18	190K	1.2M	5.2M	4.1M	1.2G

Post-hoc explanations

heuristic approaches exist

(e.g. **LIME** or **Anchor**)

heuristic approaches exist

(e.g. **LIME** or **Anchor**)

- **local** explanations

heuristic approaches exist

(e.g. **LIME** or **Anchor**)

- **local** explanations
- **no** guarantees

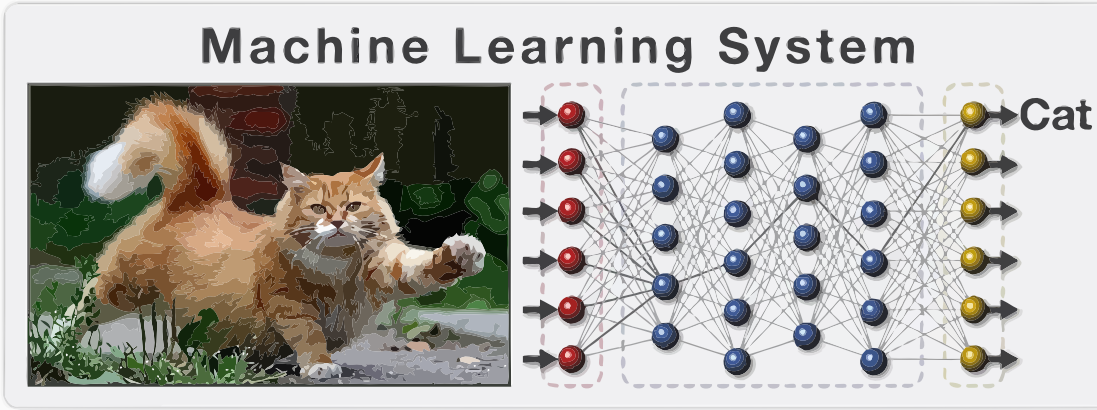
heuristic approaches exist

(e.g. **LIME** or **Anchor**)

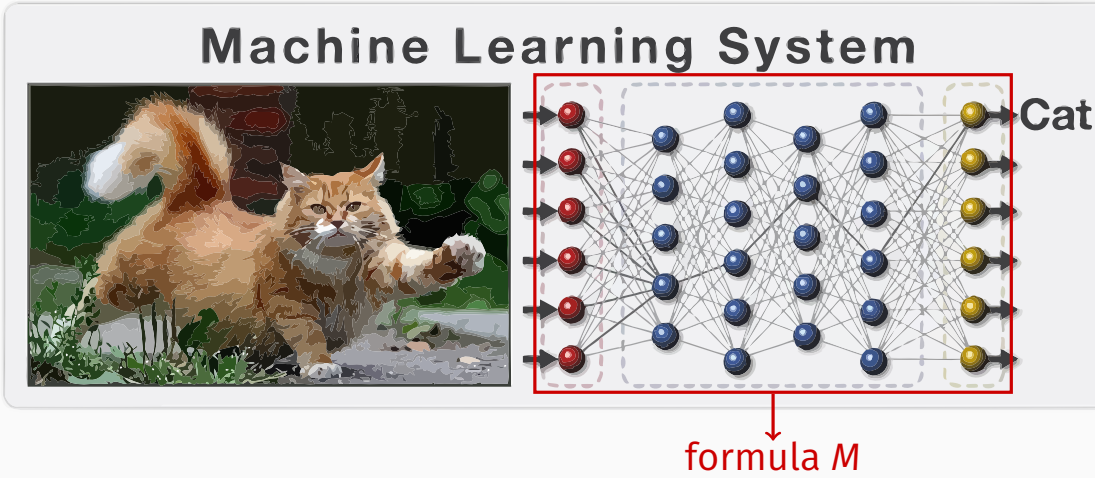
- **local** explanations
- **no** guarantees

(un-)reliable?

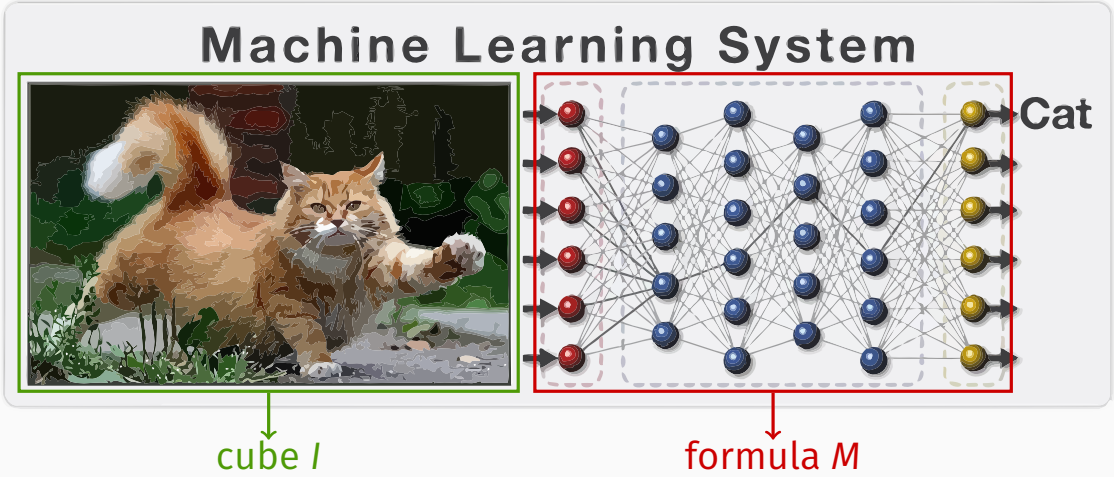
From ML model to logic



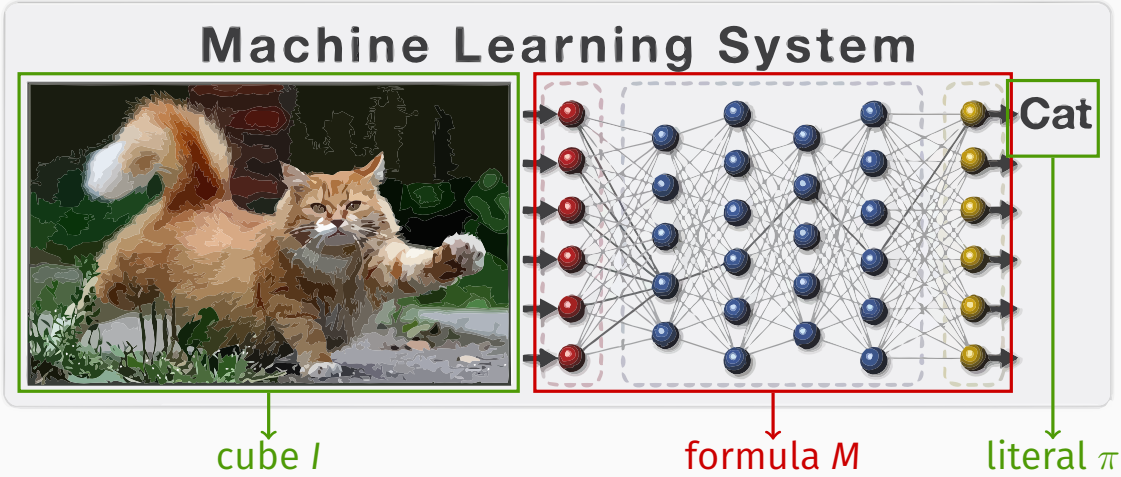
From ML model to logic

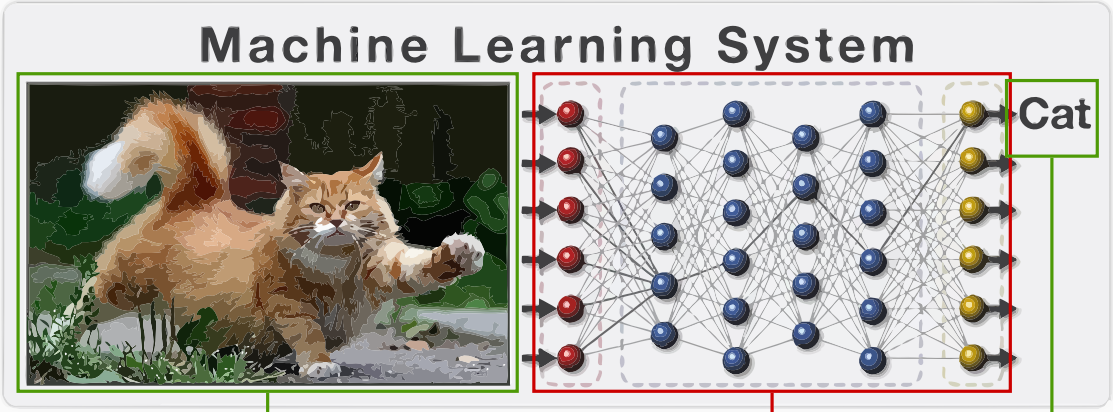


From ML model to logic



From ML model to logic





$$I \wedge M \models \pi$$

Abductive explanations of ML models

[INMS19]

given a ***classifier*** M, a ***cube*** l and a ***prediction*** π,

Abductive explanations of ML models

[INMS19]

given a ***classifier*** M, a ***cube*** l and a ***prediction*** π,

compute a (**cardinality- or subset-**) minimal $E_m \subseteq l$ s.t.

Abductive explanations of ML models

[INMS19]

given a ***classifier*** M, a ***cube*** I and a ***prediction*** π,

compute a (**cardinality- or subset-**) minimal $E_m \subseteq I$ s.t.

$$E_m \wedge M \not\models \bot$$

and

$$E_m \wedge M \models \pi$$

[INMS19]

given a ***classifier*** M, a ***cube*** I and a ***prediction*** π,

compute a (**cardinality- or subset-**) minimal $E_m \subseteq I$ s.t.

$$E_m \wedge M \not\models \bot$$

and

$$E_m \wedge M \models \pi$$

iterative **explanation procedure**

1. $E_m \wedge M \not\models \bot$

1. $E_m \wedge M \not\models \bot$ — ***tautology***

1. $E_m \wedge M \not\models \bot$ — ***tautology***
2. $E_m \wedge M \models \pi$

1. $E_m \wedge M \not\models \bot$ — ***tautology***
2. $E_m \wedge M \models \pi \iff E_m \models (M \rightarrow \pi)$

1. $E_m \wedge M \not\models \bot$ — ***tautology***
2. $E_m \wedge M \models \pi \iff E_m \models (M \rightarrow \pi)$

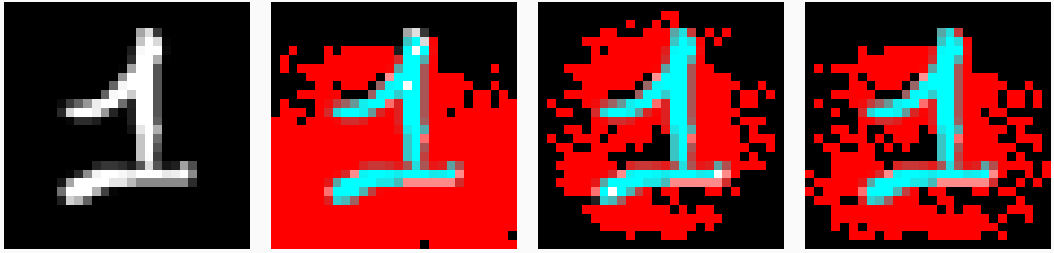
E_m is a ***prime implicant*** of $M \rightarrow \pi$

Computing one subset-minimal explanation

Input: model M, initial cube I, prediction π
Output: ***Subset-minimal*** explanation E_m

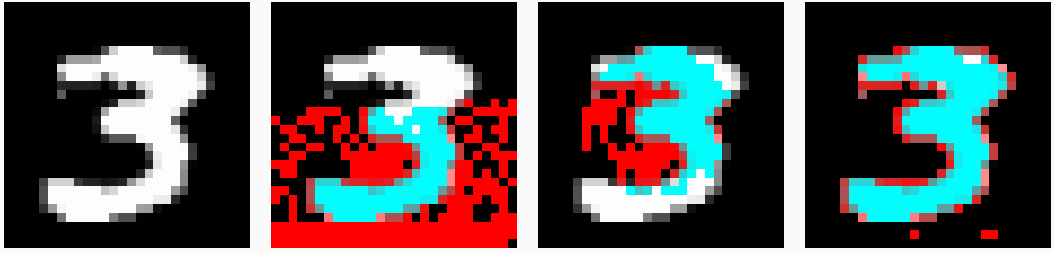
```
begin
  for l ∈ I :
    if Entails(I \ {l}, M → π) :          # make an oracle call
      I ← I \ {l}
  return I
end
```

Examples



(a) (b) (c) (d)

Figure 3: Possible minimal explanations for digit one.



(a) (b) (c) (d)

principled approach to XAI

principled approach to XAI

based on **abductive reasoning**

principled approach to XAI

based on **abductive reasoning**

applies a **reasoning oracle**, e.g. SAT, SMT or MILP

principled approach to XAI

based on **abductive reasoning**

applies a **reasoning oracle**, e.g. SAT, SMT or MILP

provides **minimality guarantees**

principled approach to XAI

based on **abductive reasoning**

applies a **reasoning oracle**, e.g. SAT, SMT or MILP

provides **minimality guarantees**

global explanations!

What next?

enumeration of **explanations**?

What next?

enumeration of **explanations**?

preferences over explanations?

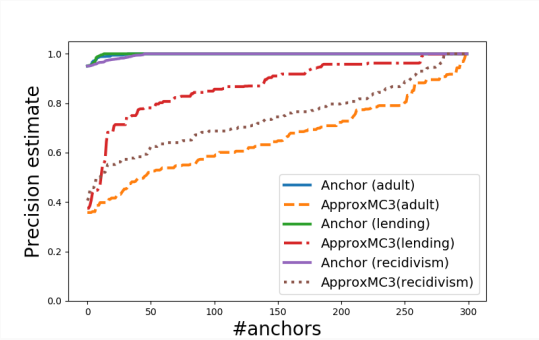
enumeration of **explanations?**

preferences over explanations?

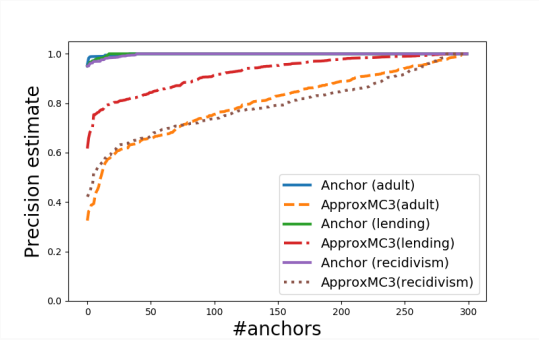
assessment heuristic approaches!

Assessing heuristic explanations

[NSM+19]



unconstrained feature space



samples with $\leq 50\%$ difference

PySAT and RC2

PySAT is a Python framework for
quick prototyping with SAT oracles

https://pysathq.github.io/

PySAT is a Python framework for
quick prototyping with SAT oracles

https://pysathq.github.io/

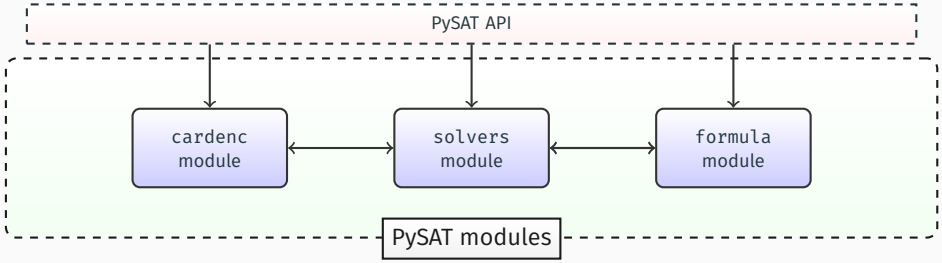
RC2 is a MaxSAT solver that
won two *complete* categories
of MaxSAT Evaluations 2018 & 2019

(FLoC Olympic Games 2018)

https://maxsat-evaluations.github.io/

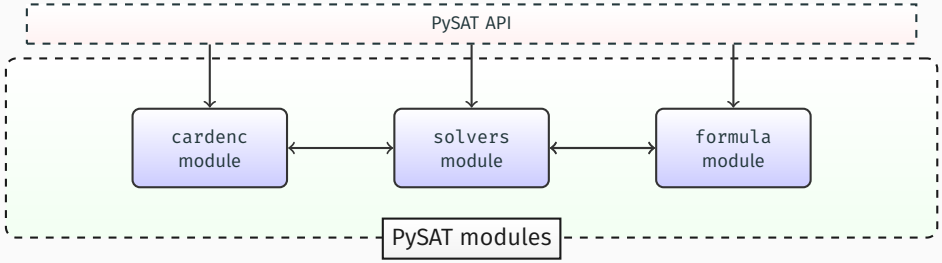
Overview of PySAT

[IMM18]



Overview of PySAT

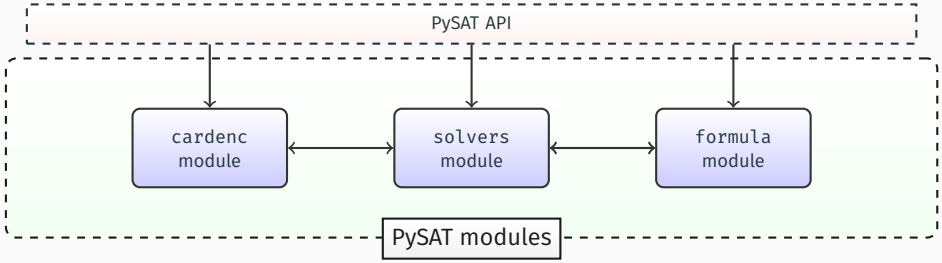
[IMM18]



- Open source, available on github

Overview of PySAT

[IMM18]



- Open source, available on github
- Comprehensive list of SAT solvers
- Comprehensive list of cardinality encodings
- Fairly comprehensive documentation
- Several use cases

Available solvers

Solver	Version
Glucose	3.0
Glucose	4.1
Lingeling	bbc-9230380-160707
Minicard	1.2
Minisat	2.2 release
Minisat	GitHub version
MapleCM	SAT competition 2018
Maplesat	MapleCOMSPS_LRB
...	...

- Solvers can either be used incrementally or non-incrementally
- Tools can use multiple solvers, e.g. for hitting set dualization or CEGAR-based QBF solving
- **URL**: `https://pysathq.github.io/docs/html/api/solvers.html`

Formula manipulation

Features
CNF & Weighted CNF (WCNF)
Read formulas from file/string
Write formulas to file
Append clauses to formula
Negate CNF formulas
Translate between CNF and WCNF
ID manager

- **URL**: `https://pysathq.github.io/docs/html/api/formula.html`

Available cardinality encodings

Name	Type
pairwise	AtMost1
bitwise	AtMost1
ladder	AtMost1
sequential counter	AtMost*k*
sorting network	AtMost*k*
cardinality network	AtMost*k*
totalizer	AtMost*k*
mtotalizer	AtMost*k*
kmtotalizer	AtMost*k*

- Also AtLeast*K* and Equals*K* constraints
- **URL**: `https://pysathq.github.io/docs/html/api/card.html`

Installation & info

- Installation:
  ```
  $ [sudo] pip2|pip3 install python-sat
  ```

- Website: https://pysathq.github.io/

Basic interface – Python3 shell

```
>>> from pysat.card import *
>>> am1 = CardEnc.atmost(lits=[1, -2, 3], encoding=EncType.pairwise)
>>> print(am1.clauses)
[[-1, 2], [-1, -3], [2, -3]]
>>>
>>> from pysat.solvers import Solver
>>> with Solver(name='m22', bootstrap_with=am1.clauses) as s:
...     if s.solve(assumptions=[1, 2, 3]) == False:
...         print(s.get_core())
[3, 1]
```

Basic interface – Python3 script

```
#!/usr/local/bin/python3
from sys import argv

from pysat.formula import CNF
from pysat.solvers import Glucose3, Solver

formula = CNF()
formula.append([-1, 2, 4])
formula.append([1, -2, 5])
formula.append([-1, -2, 6])
formula.append([1, 2, 7])

g = Glucose3(bootstrap_with=formula.clauses)

if g.solve(assumptions=[-4, -5, -6, -7]) == False:
    print("Core: ", g.get_core())
```

Example: naive (deletion) MUS extraction

Input : Set $\mathcal{F}$
Output: Minimal subset $\mathcal{M}$
begin
  $\mathcal{M} \leftarrow \mathcal{F}$
  foreach $c \in \mathcal{M}$ **do**
    if $\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // If $\neg\mathsf{SAT}(\mathcal{M} \setminus \{c\})$, then $c \notin$ MUS
  return $\mathcal{M}$ // Final $\mathcal{M}$ is MUS
end

- Number of predicate tests: $\mathcal{O}(m)$

[CD91, BDTW93]

Naive MUS extraction I

```
def main():
    cnf = CNF(from_file=argv[1])   # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)

    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)

    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)

if __name__== "__main__":
  main()
```

Naive MUS extraction II

```
def add_assumps(cnf):
    rnv = topv = cnf.nv
    assumps = []                        # list of assumptions to use
    for i in range(len(cnf.clauses)):
        topv += 1
        assumps.append(topv)            # register literal
        cnf.clauses[i].append(-topv)    # extend clause with literal
    cnf.nv = cnf.nv + len(assumps)      # update # of vars
    return rnv, assumps

def main():
    cnf = CNF(from_file=argv[1])   # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)

    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)

    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)

if __name__== "__main__":
  main()
```

Naive MUS extraction III

```
from sys import argv

from pysat.formula import CNF
from pysat.solvers import Solver


def find_mus(assmp, oracle):
    i = 0
    while i < len(assmp):
        ts = assmp[:i] + assmp[(i+1):]
        if not oracle.solve(assumptions=ts):
            assmp = ts
        else:
            i += 1
    return assmp
```

Naive MUS extraction III

```
from sys import argv

from pysat.formula import CNF
from pysat.solvers import Solver


def find_mus(assmp, oracle):
    i = 0
    while i < len(assmp):
        ts = assmp[:i] + assmp[(i+1):]
        if not oracle.solve(assumptions=ts):
            assmp = ts
        else:
            i += 1
    return assmp
```

Demo

A less naive MUS extractor

```
def clset_refine(assmp, oracle):
    assmp = sorted(assmp)
    while True:
        oracle.solve(assumptions=assmp)
        ts = sorted(oracle.get_core())
        if ts == assmp:
            break
        assmp = ts
    return assmp
# ...
def main():
    cnf = CNF(from_file=argv[1])   # create a CNF object from file
    (rnv, assumps) = add_assumps(cnf)

    oracle = Solver(name='g3', bootstrap_with=cnf.clauses)

    assumps = clset_refine(assumps, oracle)
    mus = find_mus(assumps, oracle)
    mus = [ref - rnv for ref in mus]
    print("MUS: ", mus)

if __name__== "__main__":
  main()
```

[Sur12]

given K **agents** and K **goals** on a map of **size** $N \times M$,

[Sur12]

given K **agents** and K **goals** on a map of **size** $N \times M$,

compute **“shortest path”** for all pairs (**agent**, **goal**)

[Sur12]

given K **agents** and K **goals** on a map of **size** $N \times M$,

compute **"shortest path"** for all pairs (**agent**, **goal**)

plenty of works on solving MAPF with SAT!

[Sur12]

given K **agents** and K **goals** on a map of **size** $N \times M$,

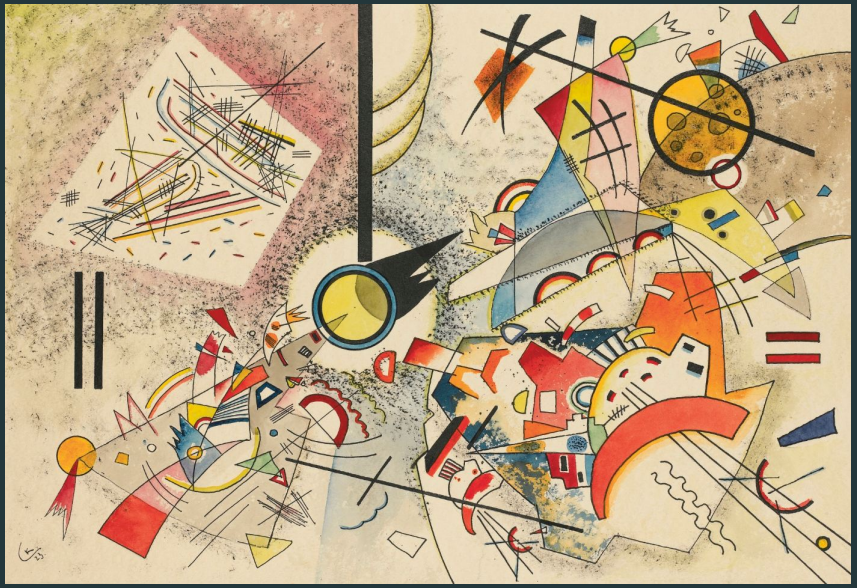
compute **"shortest path"** for all pairs (**agent**, **goal**)

plenty of works on solving MAPF with SAT!

demo for MAPF@Minecraft+PySAT:

`https://reason.di.fc.ul.pt/~aign/storage/mcmapf2.mov`

6 SAT Oracles & Proof Complexity



Proof Complexity

- CDCL is the main technique for solving SAT
- When formulas are unsatisfiable, CDCL is equivalent to Resolution
- Success of CDCL demonstrates the reach of the Resolution proof system

Proof Complexity

- CDCL is the main technique for solving SAT
- When formulas are unsatisfiable, CDCL is equivalent to Resolution
- Success of CDCL demonstrates the reach of the Resolution proof system
- Some basic problems, like pigeon-hole principle, cannot have short Resolution Refutations

Proof Complexity

- CDCL is the main technique for solving SAT
- When formulas are unsatisfiable, CDCL is equivalent to Resolution
- Success of CDCL demonstrates the reach of the Resolution proof system
- Some basic problems, like pigeon-hole principle, cannot have short Resolution Refutations
- From proof complexity point of view, Resolution is regarded as a rather weak proof system

Proof Complexity

- CDCL is the main technique for solving SAT
- When formulas are unsatisfiable, CDCL is equivalent to Resolution
- Success of CDCL demonstrates the reach of the Resolution proof system
- Some basic problems, like pigeon-hole principle, cannot have short Resolution Refutations
- From proof complexity point of view, Resolution is regarded as a rather weak proof system
- Recent efforts for developing efficient implementations of stronger proof systems:
  - Extended Resolution (ExtRes)
  - DRAT
  - Cutting Planes (CP)
  - Dual-Rail Maximum Satisfiability (DRMaxSAT)

Dual-Rail Maximum Satisfiability (DRMaxSAT)

DRMaxSAT - General Idea

- Translates a CNF formula $\mathcal{F}$ using the Dual-Rail Encoding
- Uses a MaxSAT algorithm to obtain the cost of the encoded formula
- Determines the satisfiability of $\mathcal{F}$ based on the cost of the encoded formula

Dual-Rail Encoding (DRE)

[DAC87, AI99]

Input: $\mathcal{F}$ CNF formula with N variables $X = \{x_1, \ldots, x_N\}$

Dual-Rail Encoding (DRE)

[DAC87, AI99]

Input: $\mathcal{F}$ CNF formula with N variables $X = \{x_1, \ldots, x_N\}$

Output: MaxSAT problem $< \mathcal{H}, \mathcal{S} >$:

Dual-Rail Encoding (DRE)

[DAC87, AI99]

Input: $\mathcal{F}$ CNF formula with N variables $X = \{x_1, \ldots, x_N\}$

Output: MaxSAT problem $< \mathcal{H}, \mathcal{S} >$:

- for each $x_i \in X$:
  - associate new variables p_i and n_i

$$x_i = 1 \text{ iff } p_i = 1, \text{ and } x_i = 0 \text{ iff } n_i = 1$$

  - add to $\mathcal{S}$ the clauses (p_i) and (n_i)
  - add to $\mathcal{H}$ the clause $(\neg p_i \vee \neg n_i)$ ($\mathcal{P}$ clauses)

Dual-Rail Encoding (DRE)

[DAC87, AI99]

Input: $\mathcal{F}$ CNF formula with N variables $X = \{x_1, \ldots, x_N\}$

Output: MaxSAT problem $< \mathcal{H}, \mathcal{S} >$:

- for each $x_i \in X$:
  - associate new variables p_i and n_i

$$x_i = 1 \text{ iff } p_i = 1, \text{ and } x_i = 0 \text{ iff } n_i = 1$$

  - add to $\mathcal{S}$ the clauses (p_i) and (n_i)
  - add to $\mathcal{H}$ the clause $(\neg p_i \vee \neg n_i)$ ($\mathcal{P}$ clauses)

- for each clause $c \in \mathcal{F}$ add to $\mathcal{H}$ the clause c':
  - if $x_i \in c$ then $\neg n_i \in c'$
  - if $\neg x_i \in c$ then $\neg p_i \in c'$

DRMaxSAT: DRE Example

$$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$$

DRMaxSAT: DRE Example

$$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$$

- MaxSAT problem $< \mathcal{H}, \mathcal{S} >$

DRMaxSAT: DRE Example

$$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$$

- MaxSAT problem $< \mathcal{H}, \mathcal{S} >$
- for x_1:
  - create p_1 and n_1
  - add (p_1), (n_1) to $\mathcal{S}$
  - add $(\neg p_1 \vee \neg n_1)$ to $\mathcal{H}$
- for x_2:
  - create p_2 and n_2
  - add (p_2), (n_2) to $\mathcal{S}$
  - add $(\neg p_2 \vee \neg n_2)$ to $\mathcal{H}$

DRMaxSAT: DRE Example

$$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$$

- MaxSAT problem $< \mathcal{H}, \mathcal{S} >$
- for $(\neg x_1 \vee \neg x_2)$:
  - add $(\neg p_1 \vee \neg p_2)$ to $\mathcal{H}$
- for (x_1), (x_2), $(\neg x_2)$:
  - add $(\neg n_1)$, $(\neg n_2)$, $(\neg p_2)$ to $\mathcal{H}$

DRMaxSAT: DRE Example

$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$

MaxSAT problem $< \mathcal{H}, \mathcal{S} >$

DRMaxSAT: DRE Example

$$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$$

MaxSAT problem $< \mathcal{H}, \mathcal{S} >$:

$$\mathcal{S} = \{(p_1), (n_1), (p_2), (n_1)\}$$

$$\begin{aligned}\mathcal{H} = \{&(\neg p_1 \vee \neg n_1), (\neg p_2 \vee \neg n_2),\\ &(\neg p_1 \vee \neg p_2),\\ &(\neg n_1), (\neg n_2), (\neg p_2)\}\end{aligned}$$

DRMaxSAT: DRE Example

$\mathcal{F} = \{(\neg x_1 \vee \neg x_2), (x_1), (x_2), (\neg x_2)\}$

MaxSAT problem $< \mathcal{H}, \mathcal{S} >$:

$$\mathcal{S} = \{(p_1), (n_1), (p_2), (n_1)\}$$

$$\begin{aligned}\mathcal{H} = \{&(\neg p_1 \vee \neg n_1), (\neg p_2 \vee \neg n_2),\\ &(\neg p_1 \vee \neg p_2),\\ &(\neg n_1), (\neg n_2), (\neg p_2)\}\end{aligned}$$

MaxSAT Cost: 3

DRMaxSAT: DRE Theorem

Theorem
$\mathcal{F}$ is satisfiable iff there is a truth assignment satisfying $\mathcal{H}$ that satisfies at least N clauses in $\mathcal{S}$. [SAT17]

DRMaxSAT: DRE Theorem

Theorem
$\mathcal{F}$ is satisfiable iff there is a truth assignment satisfying $\mathcal{H}$ that satisfies at least N clauses in $\mathcal{S}$. [SAT17]

Example: $N = 2$ and MaxSAT cost 3, thus $\mathcal{F}$ is unsatisfiable.

DRMaxSAT: General Pseudo-Code

input: $\mathcal{F}$

$\mathsf{HEnc}(\mathcal{F}) = \langle \mathcal{H}, \mathcal{S} \rangle \leftarrow \mathsf{DualRailEncode}(\mathcal{F})$

cost $\leftarrow \mathsf{ApplyMaxSAT}(\mathsf{HEnc}(\mathcal{F}))$

DRMaxSAT: General Pseudo-Code

input: $\mathcal{F}$

$\mathsf{HEnc}(\mathcal{F}) = \langle \mathcal{H}, \mathcal{S} \rangle \leftarrow \mathsf{DualRailEncode}(\mathcal{F})$

cost $\leftarrow \mathsf{ApplyMaxSAT}(\mathsf{HEnc}(\mathcal{F}))$

if cost $\leq |\mathsf{var}(\mathcal{F})|$ **then**
  return *true*
end

DRMaxSAT: General Pseudo-Code

input: $\mathcal{F}$

$\mathsf{HEnc}(\mathcal{F}) = \langle \mathcal{H}, \mathcal{S} \rangle \leftarrow \mathsf{DualRailEncode}(\mathcal{F})$

cost $\leftarrow \mathsf{ApplyMaxSAT}(\mathsf{HEnc}(\mathcal{F}))$

if cost $\leq |\mathsf{var}(\mathcal{F})|$ **then**
  return *true*
end
else
  return *false*
end

DRMaxSAT: Theoretical Results

What is the power of DRMaxSAT?

DRMaxSAT: Theoretical Results

What is the power of DRMaxSAT?

Considered three MaxSAT approaches with DRMaxSAT:

- MaxSAT Resolution
- Core-guided MaxSAT Algorithms
- Implict Hitting Set MaxSAT Algorithms (MaxHS-like algorithms)

DRMaxSAT: Summary of Theoretical Results on MaxSAT Resolution

Theorem
Multiple DRMaxSAT simulates tree-like Resolution. [Source: AAAI18]

Theorem
Weighted DRMaxSAT simulates general Resolution. [Source: AAAI18]

DRMaxSAT: Summary of Theoretical Results on MaxSAT Resolution

Theorem
Multiple DRMaxSAT simulates tree-like Resolution. [Source: AAAI18]

Theorem
Weighted DRMaxSAT simulates general Resolution. [Source: AAAI18]

Theorem
DRMaxSAT refutes both PHP and 2PHP in polynomial time. [Source: SAT17,AAAI18]

DRMaxSAT: Summary of Theoretical Results on MaxSAT Resolution

Theorem
Multiple DRMaxSAT simulates tree-like Resolution. [Source: AAAI18]

Theorem
Weighted DRMaxSAT simulates general Resolution. [Source: AAAI18]

Theorem
DRMaxSAT refutes both PHP and 2PHP in polynomial time. [Source: SAT17,AAAI18]

Theorem
The DRMaxSAT proof system does not polynomially simulate CP. [Source: AAAI18]

DRMaxSAT: Summary of Theoretical Results on MaxSAT Resolution

Theorem
Multiple DRMaxSAT simulates tree-like Resolution. [Source: AAAI18]

Theorem
Weighted DRMaxSAT simulates general Resolution. [Source: AAAI18]

Theorem
DRMaxSAT refutes both PHP and 2PHP in polynomial time. [Source: SAT17,AAAI18]

Theorem
The DRMaxSAT proof system does not polynomially simulate CP. [Source: AAAI18]

- DRMaxSAT is strictly stronger than Resolution
- DRMaxSAT does not simulate Cutting Planes

DRMaxSAT: Summary of Theoretical Results with Core-guided Algorithms

Theorem
Core-guided MaxSAT with the dual-rail encoding p-simulates unrestricted Resolution.

DRMaxSAT: Summary of Theoretical Results with Core-guided Algorithms

Theorem
Core-guided MaxSAT with the dual-rail encoding p-simulates unrestricted Resolution.

Theorem
Core-guided MaxSAT with the dual-rail encoding refutes both PHP and 2PHP in polynomial time.

[Source: SAT17,AAAI18]

DRMaxSAT: Summary of Theoretical Results with Core-guided Algorithms

Theorem
Core-guided MaxSAT with the dual-rail encoding p-simulates unrestricted Resolution.

Theorem
Core-guided MaxSAT with the dual-rail encoding refutes both PHP and 2PHP in polynomial time.

[Source: SAT17,AAAI18]

- Core-guided MaxSAT with the dual-rail encoding is strictly stronger than Resolution

DRMaxSAT: Summary of Theoretical Results with MaxHS-like Algorithms

Theorem
MaxHS-like MaxSAT Algorithms with the dual-rail encoding refutes both PHP and 2PHP in polynomial time. *[Source: SAT19]*

DRMaxSAT: Experimental Results

What is the behaviour of DRMaxSAT in practice?

DRMaxSAT: Experimental Results

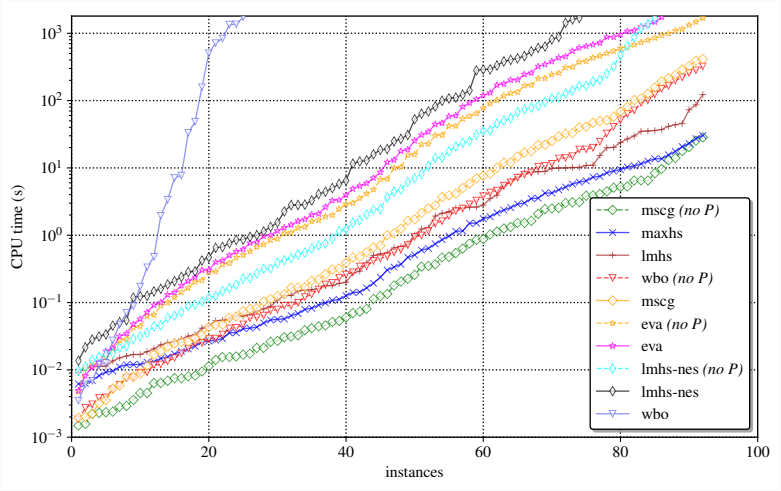
What is the behaviour of DRMaxSAT in practice?

Several problems/principles hard for resolution:

- (PHP) Pigeonhole Principle
- (2PHP) Doubled Pigeonhole Principle
- (MCB) Mutilated Chessboard Principle
- (URQ) Urquhart/Tseitin formulas
- (COMB) Combination formulas $\text{PHP}_m^{m+1} \vee \text{URQ}_{n,i}$

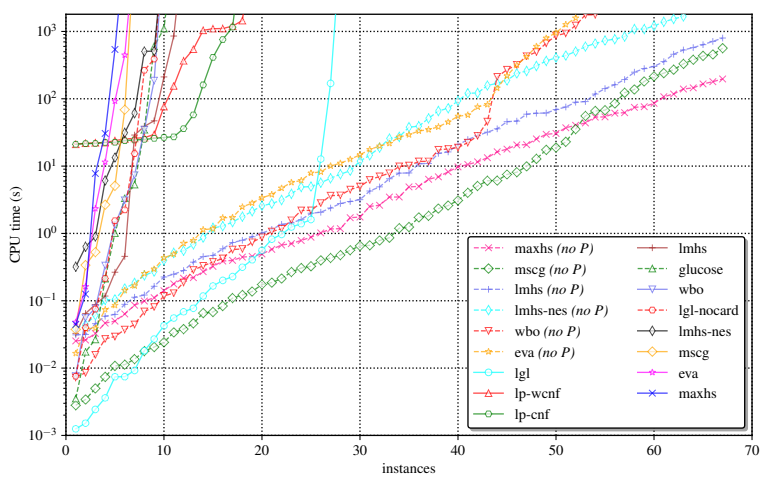
DRMaxSAT: Experimental Results - PHP

[Source: SAT17]



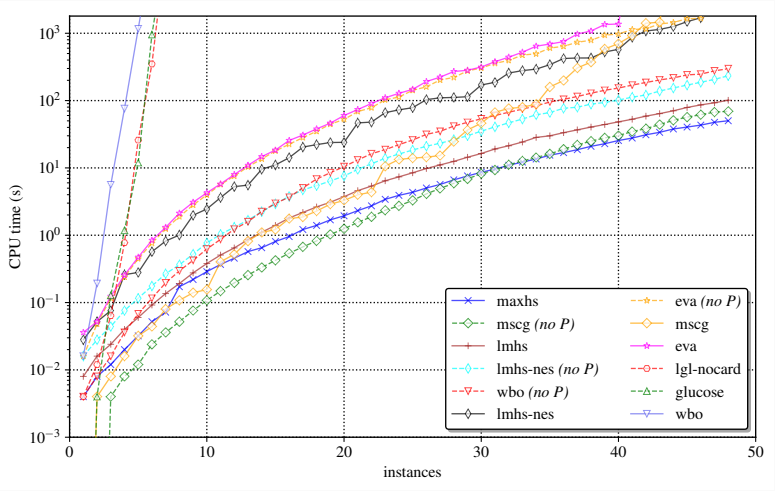
DRMaxSAT: Experimental Results - 2PHP

[Source: AAAI18]



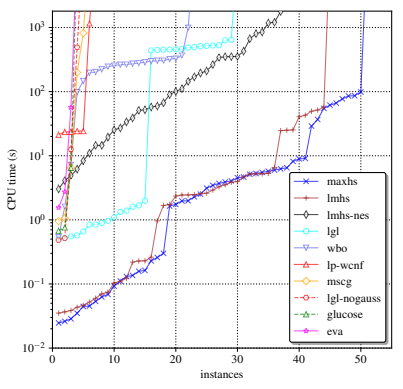
DRMaxSAT: Experimental Results - MCB

[Source: AAAI18]

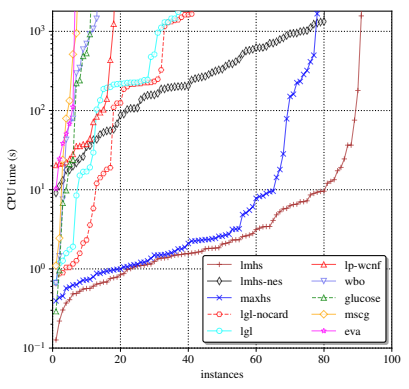


DRMaxSAT: Experimental Results - URQ & COMB

[Source: SAT17]



URQ



COMB

7 A Glimpse of the Future

Take home messages

- Remarkable improvements in (CDCL) SAT solver performance
  - SAT is a low-level, but very powerful problem solving paradigm

Take home messages

- Remarkable improvements in (CDCL) SAT solver performance
  - SAT is a low-level, but very powerful problem solving paradigm
- Wide range of applications of SAT oracles
  - High profile applications, e.g. verification of ML models & XAI
  - Solving problems (well) beyond NP
  - Recent inroads in proof complexity
    - Ongoing arms race for proof systems stronger than resolution/clause learning, e.g. extended resolution, cutting planes and DRMaxSAT

Take home messages

- Remarkable improvements in (CDCL) SAT solver performance
  - SAT is a low-level, but very powerful problem solving paradigm
- Wide range of applications of SAT oracles
  - High profile applications, e.g. verification of ML models & XAI
  - Solving problems (well) beyond NP
  - Recent inroads in proof complexity
    - Ongoing arms race for proof systems stronger than resolution/clause learning, e.g. extended resolution, cutting planes and DRMaxSAT
- Also, there are other success stories, e.g. ILP

Some final notes

- There is an ongoing revolution on problem solving with SAT oracles
  - E.g. QBF, model-based diagnosis, explainability, theorem proving, program synthesis, XAI, ...
  - But also with ILP & SMT oracles
- The use of SAT/ILP/SMT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.
  - More to be expected
    - E.g. the age of **modular reasoning**?

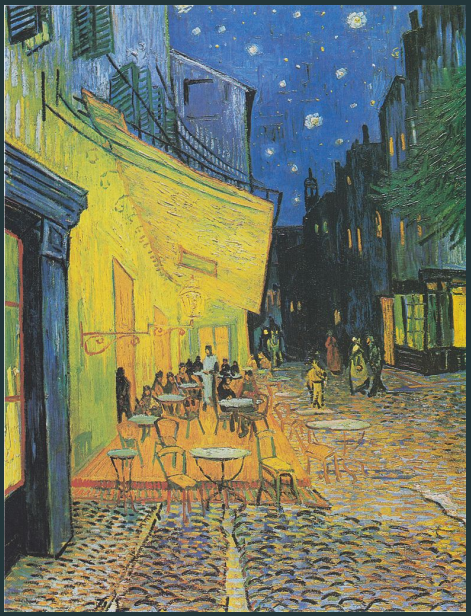
Some final notes

- There is an ongoing revolution on problem solving with SAT oracles
  - E.g. QBF, model-based diagnosis, explainability, theorem proving, program synthesis, XAI, ...
  - But also with ILP & SMT oracles
- The use of SAT/ILP/SMT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.
  - More to be expected
    - E.g. the age of **modular reasoning**?
- Many fascinating oracle-related research topics out there **!**
  - Additional connections with rigorous reasoning in ML expected

Sample of tools

- PySAT
- SAT solvers:
  - MiniSat
  - Glucose
- MaxSAT solvers:
  - RC2
  - MSCG
  - OpenWBO
  - MaxHS
- MUS extractors:
  - MUSer
- MCS extractors:
  - mcsXL
  - LBX
  - MCSls
- Many other tools available from the ReasonLab server

Questions?



References i

[ABGL12] Carlos Ansótegui, Maria Luisa Bonet, Joel Gabàs, and Jordi Levy.
Improving SAT-based weighted MaxSAT solvers.
In *CP*, pages 86–101, 2012.

[ABL09] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
Solving (weighted) partial MaxSAT through satisfiability testing.
In *SAT*, pages 427–440, 2009.

[ABL10a] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
A new algorithm for weighted partial MaxSAT.
In *AAAI*, 2010.

[ABL+10b] Josep Argelich, Daniel Le Berre, Inês Lynce, Joao Marques-Silva, and Pascal Rapicault.
Solving linux upgradeability problems using boolean optimization.
In *LoCoCo*, volume 29 of *EPTCS*, pages 11–22, 2010.

[ABL13] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy.
SAT-based MaxSAT algorithms.
Artif. Intell., 196:77–105, 2013.

[AL08] Josep Argelich and Inês Lynce.
CNF instances from the software package installation problem.
In *RCRA*, volume 451 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

References ii

[ALS09] Josep Argelich, Inês Lynce, and João P. Marques Silva.
On solving boolean multilevel optimization problems.
In *IJCAI*, pages 393–398, 2009.

[AMM15] M. Fareed Arif, Carlos Mencía, and Joao Marques-Silva.
Efficient MUS enumeration of horn formulae with applications to axiom pinpointing.
In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 324–342. Springer, 2015.

[ANO+12] Ignasi Abío, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Valentin Mayer-Eichberger.
A new look at BDDs for pseudo-boolean constraints.
J. Artif. Intell. Res., 45:443–480, 2012.

[ANOR09] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell.
Cardinality networks and their applications.
In *SAT*, pages 167–180, 2009.

[ANOR11] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell.
Cardinality networks: a theoretical and empirical study.
Constraints, 16(2):195–221, 2011.

[AS09] Gilles Audemard and Laurent Simon.
Predicting learnt clauses quality in modern SAT solvers.
In *IJCAI*, pages 399–404, 2009.

References iii

[Bat68] Kenneth E. Batcher.
Sorting networks and their applications.
In *AFIPS Spring Joint Computing Conference*, volume 32 of *AFIPS Conference Proceedings*, pages 307–314. Thomson Book Company, Washington D.C., 1968.

[BBR09] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel.
New encodings of pseudo-boolean constraints into CNF.
In *SAT*, pages 181–194, 2009.

[BDTW93] R. R. Bakker, F. Dikker, F. Tempelman, and P. M. Wognum.
Diagnosing and solving over-determined constraint satisfaction problems.
In *IJCAI*, pages 276–281, 1993.

[BF15] Armin Biere and Andreas Fröhlich.
Evaluating CDCL restart schemes.
In *Sixth Pragmatics of SAT workshop*, 2015.

[Bie08] Armin Biere.
PicoSAT essentials.
JSAT, 4(2-4):75–97, 2008.

[BK15] Fahiem Bacchus and George Katsirelos.
Using minimal correction sets to more efficiently compute minimal unsatisfiable sets.
In *CAV (2)*, volume 9207 of *Lecture Notes in Computer Science*, pages 70–86. Springer, 2015.

References iv

[BKS04] Paul Beame, Henry A. Kautz, and Ashish Sabharwal.
Towards understanding and harnessing the potential of clause learning.
J. Artif. Intell. Res., 22:319–351, 2004.

[BLM12] Anton Belov, Inês Lynce, and Joao Marques-Silva.
Towards efficient MUS extraction.
AI Commun., 25(2):97–116, 2012.

[BMS00] Luís Baptista and Joao Marques-Silva.
Using randomization and learning to solve hard real-world instances of satisfiability.
In *CP*, volume 1894 of *Lecture Notes in Computer Science*, pages 489–494. Springer, 2000.

[BP10] Daniel Le Berre and Anne Parrain.
The Sat4j library, release 2.2.
JSAT, 7(2-3):59–6, 2010.

[BS05] James Bailey and Peter J. Stuckey.
Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization.
In *PADL*, pages 174–186, 2005.

[CD91] John W. Chinneck and Erik W. Dravnieks.
Locating minimal infeasible constraint sets in linear programs.
INFORMS Journal on Computing, 3(2):157–168, 1991.

References v

[Coo71] Stephen A. Cook.
The complexity of theorem-proving procedures.
In *STOC*, pages 151–158. ACM, 1971.

[CT95] Zhi-Zhong Chen and Seinosuke Toda.
The complexity of selecting maximal solutions.
Inf. Comput., 119(2):231–239, 1995.

[CZ10] Michael Codish and Moshe Zazon-Ivry.
Pairwise cardinality networks.
In *LPAR (Dakar)*, volume 6355 of *Lecture Notes in Computer Science*, pages 154–172. Springer, 2010.

[DB11] Jessica Davies and Fahiem Bacchus.
Solving MAXSAT by solving a sequence of simpler SAT instances.
In *CP*, pages 225–239, 2011.

[DB13a] Jessica Davies and Fahiem Bacchus.
Exploiting the power of MIP solvers in MAXSAT.
In *SAT*, pages 166–181, 2013.

[DB13b] Jessica Davies and Fahiem Bacchus.
Postponing optimization to speed up MAXSAT solving.
In *CP*, pages 247–262, 2013.

References vi

[dK89] Johan de Kleer.
A comparison of ATMS and CSP techniques.
In *IJCAI*, pages 290–296. Morgan Kaufmann, 1989.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland.
A machine program for theorem-proving.
Commun. ACM, 5(7):394–397, 1962.

[DP60] Martin Davis and Hilary Putnam.
A computing procedure for quantification theory.
J. ACM, 7(3):201–215, 1960.

[dSNP88] J. L. de Siqueira N. and Jean-Francois Puget.
Explanation-based generalisation of failures.
In *ECAI*, pages 339–344, 1988.

[ES03] Niklas Eén and Niklas Sörensson.
An extensible SAT-solver.
In *SAT*, pages 502–518, 2003.

[ES06] Niklas Eén and Niklas Sörensson.
Translating pseudo-boolean constraints into SAT.
JSAT, 2(1-4):1–26, 2006.

References vii

[FM06] Zhaohui Fu and Sharad Malik.
On solving the partial MAX-SAT problem.
In *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, 2006.

[FP01] Alan M. Frisch and Timothy J. Peugniez.
Solving non-boolean satisfiability problems with stochastic local search.
In *IJCAI*, pages 282–290. Morgan Kaufmann, 2001.

[FS02] Torsten Fahle and Meinolf Sellmann.
Cost based filtering for the constrained knapsack problem.
Annals OR, 115(1-4):73–93, 2002.

[Gav07] Marco Gavanelli.
The log-support encoding of CSP into SAT.
In *CP*, volume 4741 of *Lecture Notes in Computer Science*, pages 815–822. Springer, 2007.

[Gel09] Allen Van Gelder.
Improved conflict-clause minimization leads to improved propositional proof traces.
In *SAT*, pages 141–146, 2009.

[Gen02] Ian P. Gent.
Arc consistency in SAT.
In *ECAI*, pages 121–125. IOS Press, 2002.

References viii

[GF93] Georg Gottlob and Christian G. Fermüller.
Removing redundancy from a clause.
Artif. Intell., 61(2):263–289, 1993.

[GJ96] Richard Génisson and Philippe Jégou.
Davis and putnam were already checking forward.
In *ECAI*, pages 180–184, 1996.

[GN02] Evguenii I. Goldberg and Yakov Novikov.
BerkMin: A fast and robust SAT-solver.
In *DATE*, pages 142–149. IEEE Computer Society, 2002.

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato.
Heavy-tailed distributions in combinatorial search.
In *CP*, volume 1330 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 1997.

[HJL+15] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere.
Clause elimination for SAT and QSAT.
J. Artif. Intell. Res., 53:127–168, 2015.

[HLSB06] Fred Hemery, Christophe Lecoutre, Lakhdar Sais, and Frédéric Boussemart.
Extracting MUCs from constraint networks.
In *ECAI*, pages 113–117, 2006.

References ix

[HMM11] Federico Heras, António Morgado, and Joao Marques-Silva.
Core-guided binary search algorithms for maximum satisfiability.
In *AAAI*. AAAI Press, 2011.

[Hua07] Jinbo Huang.
The effect of restarts on the efficiency of clause learning.
In *IJCAI*, pages 2318–2323, 2007.

[IJM14] Alexey Ignatiev, Mikolas Janota, and Joao Marques-Silva.
Towards efficient optimization in package management systems.
In *ICSE*, pages 745–755, 2014.

[IMM+14] Alexey Ignatiev, António Morgado, Vasco M. Manquinho, Inês Lynce, and Joao Marques-Silva.
Progression in maximum satisfiability.
In *ECAI*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 453–458. IOS Press, 2014.

[IMM16] Alexey Ignatiev, António Morgado, and Joao Marques-Silva.
Propositional abduction with implicit hitting sets.
In *ECAI*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 1327–1335. IOS Press, 2016.

[IMM17] Alexey Ignatiev, António Morgado, and Joao Marques-Silva.
Cardinality encodings for graph optimization problems.
In *IJCAI*, pages 652–658, 2017.

References x

[IMM18] Alexey Ignatiev, António Morgado, and Joao Marques-Silva.
PySAT: A python toolkit for prototyping with SAT oracles.
In *SAT*, volume 10929 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 2018.

[IMWM19] Alexey Ignatiev, Antonio Morgado, Georg Weissenbacher, and Joao Marques-Silva.
Model-based diagnosis with multiple observations.
In *IJCAI*, pages 1108–1115, 2019.

[INMS19] Alexey Ignatiev, Nina Narodytska, and Joao Marques-Silva.
Abduction-based explanations for machine learning models.
In *AAAI*, 2019.

[IPM15] Alexey Ignatiev, Alessandro Previti, and Joao Marques-Silva.
SAT-based formula simplification.
In *SAT*, volume 9340 of *Lecture Notes in Computer Science*, pages 287–298. Springer, 2015.

[IPNM18] Alexey Ignatiev, Filipe Pereira, Nina Narodytska, and João Marques-Silva.
A SAT-based approach to learn explainable decision sets.
In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018.

[JHB12] Matti Järvisalo, Marijn Heule, and Armin Biere.
Inprocessing rules.
In *IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

References xi

[Jun04] Ulrich Junker.
QUICKXPLAIN: preferred explanations and relaxations for over-constrained problems.
In *AAAI*, pages 167–172, 2004.

[Kas90] Simon Kasif.
On the parallel complexity of discrete relaxation in constraint satisfaction networks.
Artif. Intell., 45(3):275–286, 1990.

[LGPC16a] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki.
Exponential recency weighted average branching heuristic for SAT solvers.
In *AAAI*, pages 3434–3440, 2016.

[LGPC16b] Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki.
Learning rate based branching heuristic for SAT solvers.
In *SAT*, pages 123–140, 2016.

[LLX+17] Mao Luo, Chu-Min Li, Fan Xiao, Felip Manyà, and Zhipeng Lü.
An effective learnt clause minimization approach for CDCL SAT solvers.
In *IJCAI*, pages 703–711, 2017.

[LOM+18] Jia Hui Liang, Chanseok Oh, Minu Mathew, Ciza Thomas, Chunxiao Li, and Vijay Ganesh.
Machine learning-based restart policy for CDCL SAT solvers.
In *SAT*, pages 94–110, 2018.

References xii

[LPMM16] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva.
Fast, flexible MUS enumeration.
Constraints, 21(2):223–250, 2016.

[LS08] Mark H. Liffiton and Karem A. Sakallah.
Algorithms for computing minimal unsatisfiable subsets of constraints.
J. Autom. Reasoning, 40(1):1–33, 2008.

[MBC+06] Fabio Mancinelli, Jaap Boender, Roberto Di Cosmo, Jerome Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen.
Managing the complexity of large free and open source package-based software distributions.
In *ASE*, pages 199–208, 2006.

[MDM14] António Morgado, Carmine Dodaro, and Joao Marques-Silva.
Core-guided MaxSAT with soft cardinality constraints.
In *CP*, volume 8656 of *Lecture Notes in Computer Science*, pages 564–573. Springer, 2014.

[MHM12] António Morgado, Federico Heras, and João Marques-Silva.
Improvements to core-guided binary search for MaxSAT.
In *SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 284–297. Springer, 2012.

[MIM14] António Morgado, Alexey Ignatiev, and João Marques-Silva.
MSCG: robust core-guided MaxSAT solving.
JSAT, 9:129–134, 2014.

References xiii

[MJB13] Joao Marques-Silva, Mikolás Janota, and Anton Belov.
Minimal sets over monotone predicates in boolean formulae.
In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 592–607. Springer, 2013.

[MJIM15] Joao Marques-Silva, Mikolás Janota, Alexey Ignatiev, and António Morgado.
Efficient model based diagnosis with maximum satisfiability.
In *IJCAI*, pages 1966–1972. AAAI Press, 2015.

[MMSP09] Vasco M. Manquinho, Joao Marques-Silva, and Jordi Planes.
Algorithms for weighted boolean optimization.
In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 495–508. Springer, 2009.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik.
Chaff: Engineering an efficient SAT solver.
In *DAC*, pages 530–535. ACM, 2001.

[MP08] Joao Marques-Silva and Jordi Planes.
Algorithms for maximum satisfiability using unsatisfiable cores.
In *DATE*, pages 408–413. ACM, 2008.

[MS95] J. Marques-Silva.
Search Algorithms for Satisfiability Problems in Combinational Switching Circuits.
PhD thesis, University of Michigan, May 1995.

References xiv

[MSL11] Joao Marques-Silva and Inês Lynce.
On improving MUS extraction algorithms.
In *SAT*, volume 6695 of *Lecture Notes in Computer Science*, pages 159–173. Springer, 2011.

[MSS93] Joao Marques-Silva and Karem A. Sakallah.
Space pruning heuristics for path sensitization in test pattern generation.
Technical Report CSE-TR-178-93, University of Michigan, 1993.

[MSS94] Joao Marques-Silva and Karem A. Sakallah.
Dynamic search-space pruning techniques in path sensitization.
In *DAC*, pages 705–711. ACM Press, 1994.

[MSS96a] Joao Marques-Silva and Karem A. Sakallah.
Conflict analysis in search algorithms for propositional satisfiability.
Technical Report RT-04-96, INESC, May 1996.

[MSS96b] Joao Marques-Silva and Karem A. Sakallah.
GRASP - a new search algorithm for satisfiability.
In *ICCAD*, pages 220–227, 1996.

[MSS99] Joao Marques-Silva and Karem A. Sakallah.
GRASP: A search algorithm for propositional satisfiability.
IEEE Trans. Computers, 48(5):506–521, 1999.

References xv

[NB14] Nina Narodytska and Fahiem Bacchus.
Maximum satisfiability using core-guided maxsat resolution.
In *AAAI*, pages 2717–2723. AAAI Press, 2014.

[NIPM18] Nina Narodytska, Alexey Ignatiev, Filipe Pereira, and Joao Marques-Silva.
Learning optimal decision trees with SAT.
In *IJCAI*, pages 1362–1368, 2018.

[NSM+19] Nina Narodytska, Aditya A. Shrotri, Kuldeep S. Meel, Alexey Ignatiev, and Joao Marques-Silva.
Assessing heuristic machine learning explanations with model counting.
In *SAT*, pages 267–278, 2019.

[PD07] Knot Pipatsrisawat and Adnan Darwiche.
A lightweight component caching scheme for satisfiability solvers.
In *SAT*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, 2007.

[PD09] Knot Pipatsrisawat and Adnan Darwiche.
On the power of clause-learning SAT solvers with restarts.
In *CP*, volume 5732 of *Lecture Notes in Computer Science*, pages 654–668. Springer, 2009.

[PD11] Knot Pipatsrisawat and Adnan Darwiche.
On the power of clause-learning SAT solvers as resolution engines.
Artif. Intell., 175(2):512–525, 2011.

References xvi

[PG86] David A. Plaisted and Steven Greenbaum.
A structure-preserving clause form translation.
J. Symb. Comput., 2(3):293–304, 1986.

[Pre07] Steven David Prestwich.
Variable dependency in local search: Prevention is better than cure.
In *SAT*, pages 107–120, 2007.

[Rei87] Raymond Reiter.
A theory of diagnosis from first principles.
Artif. Intell., 32(1):57–95, 1987.

[Rob65] John Alan Robinson.
A machine-oriented logic based on the resolution principle.
J. ACM, 12(1):23–41, 1965.

[SB09] Niklas Sörensson and Armin Biere.
Minimizing learned clauses.
In *SAT*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer, 2009.

[Sel03] Meinolf Sellmann.
Approximated consistency for knapsack constraints.
In *CP*, pages 679–693, 2003.

References xvii

[Sin05] Carsten Sinz.
Towards an optimal CNF encoding of boolean cardinality constraints.
In *CP*, pages 827–831, 2005.

[SMV+07] Sean Safarpour, Hratch Mangassarian, Andreas G. Veneris, Mark H. Liffiton, and Karem A. Sakallah.
Improved design debugging using maximum satisfiability.
In *FMCAD*, pages 13–19. IEEE Computer Society, 2007.

[SSS12] Ashish Sabharwal, Horst Samulowitz, and Meinolf Sellmann.
Learning back-clauses in SAT.
In *SAT*, pages 498–499, 2012.

[Stu13] Peter J. Stuckey.
There are no CNF problems.
In *SAT*, pages 19–21, 2013.

[Sur12] Pavel Surynek.
A sat-based approach to cooperative path-finding using all-different constraints.
In *SOCS*, 2012.

[SZGN17] Xujie Si, Xin Zhang, Radu Grigore, and Mayur Naik.
Maximum satisfiability in software analysis: Applications and techniques.
In *CAV*, pages 68–94, 2017.

References xviii

[Tri03] Michael A. Trick.
A dynamic programming approach for consistency and propagation for knapsack constraints.
Annals OR, 118(1-4):73–84, 2003.

[Tse68] G.S. Tseitin.
On the complexity of derivations in the propositional calculus.
In H.A.O. Slesenko, editor, *Structures in Constructives Mathematics and Mathematical Logic, Part II*, pages 115–125, 1968.

[TSJL07] Chris Tucker, David Shuffelton, Ranjit Jhala, and Sorin Lerner.
OPIUM: optimal package install/uninstall manager.
In *ICSE*, pages 178–188, 2007.

[TTKB09] Naoyuki Tamura, Akiko Taga, Satoshi Kitagawa, and Mutsunori Banbara.
Compiling finite linear CSP into SAT.
Constraints, 14(2):254–272, 2009.

[vMW08] Hans van Maaren and Siert Wieringa.
Finding guaranteed MUSes fast.
In *SAT*, pages 291–304, 2008.

[Wal00] Toby Walsh.
SAT v CSP.
In *CP*, volume 1894 of *Lecture Notes in Computer Science*, pages 441–456. Springer, 2000.

[War98] Joost P. Warners.
A linear-time transformation of linear inequalities into conjunctive normal form.
Inf. Process. Lett., 68(2):63–69, 1998.

[ZM03] Lintao Zhang and Sharad Malik.
Validating SAT solvers using an independent resolution-based checker: Practical implementations and other applications.
In *DATE*, pages 10880–10885. IEEE Computer Society, 2003.

[ZMMM01] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik.
Efficient conflict driven learning in boolean satisfiability solver.
In *ICCAD*, pages 279–285. IEEE Computer Society, 2001.

[ZS00] Hantao Zhang and Mark E. Stickel.
Implementing the Davis-Putnam method.
J. Autom. Reasoning, 24(1/2):277–296, 2000.