SAT Tutorial

Joao Marques-Silva

University of Lisbon, Portugal

First Indian SAT+SMT School

TIFR, Mumbai, India

December 04-10 2016

The CDCL SAT disruption

- SAT is NP-complete [C71]

The CDCL SAT disruption

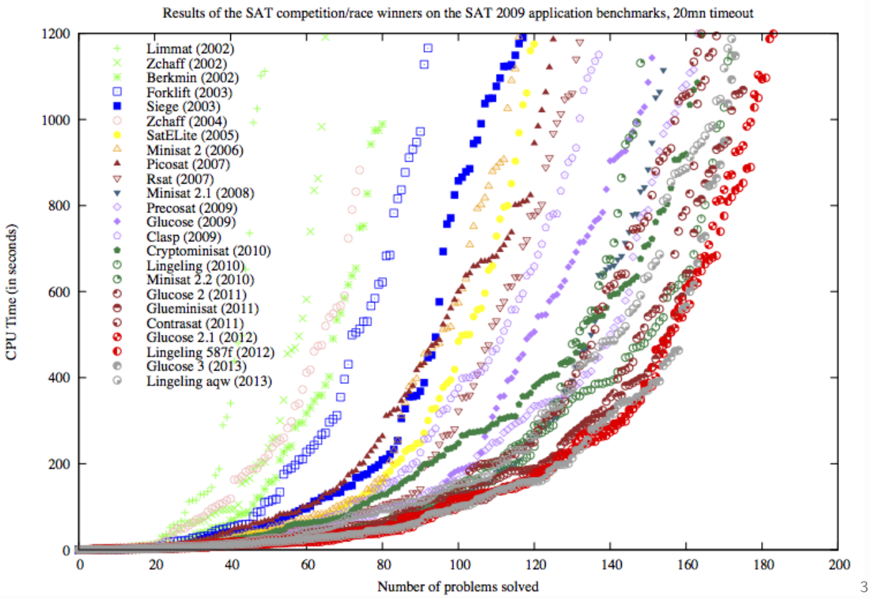
- SAT is NP-complete [C71]
  - But, CDCL SAT solving is a success story of Computer Science

The CDCL SAT disruption

- SAT is NP-complete [C71]
  - But, CDCL SAT solving is a success story of Computer Science
  - CDCL SAT solving has been truly disruptive
  - Hundreds (thousands?) of practical applications

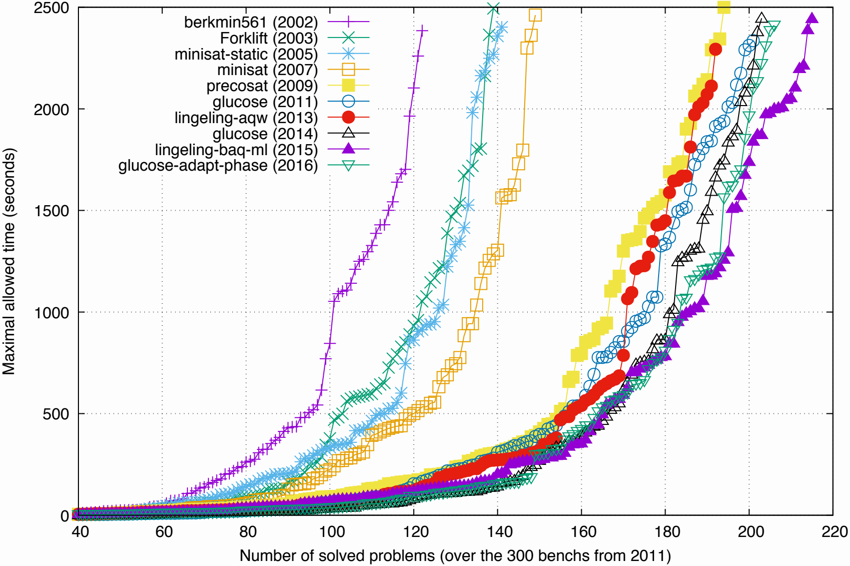
CDCL SAT solver improvement I

[Source: Le Berre 2013]

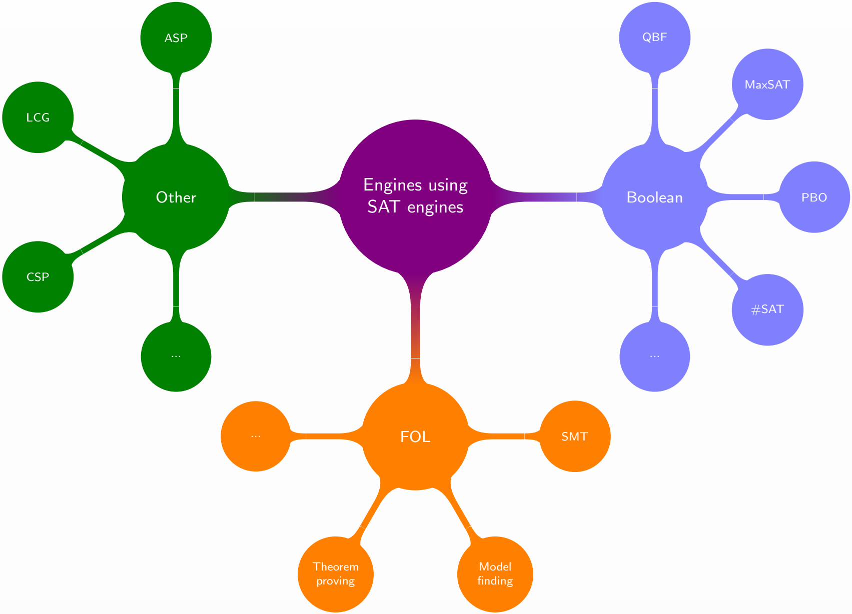


CDCL SAT solver improvement II

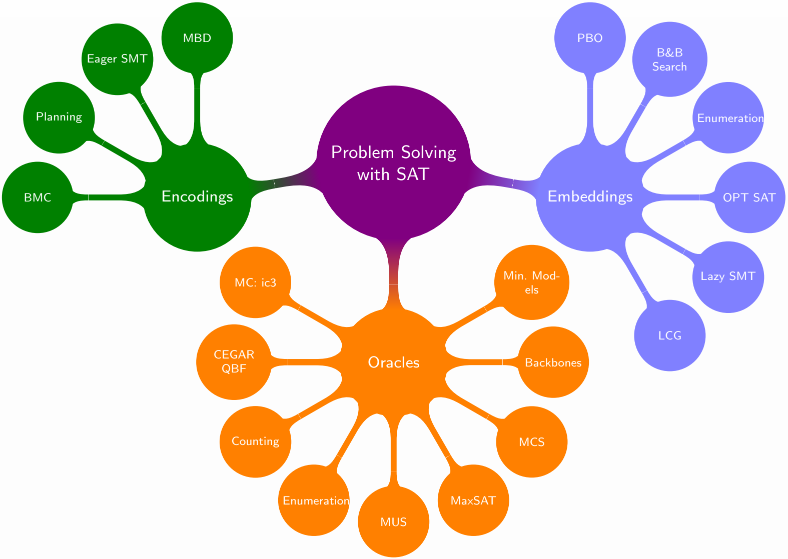
[Source: Simon 2015]



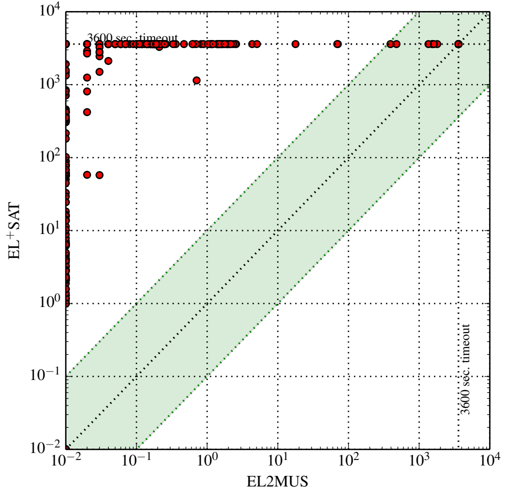
CDCL SAT is **the** engines' engine



CDCL SAT is ubiquitous in problem solving

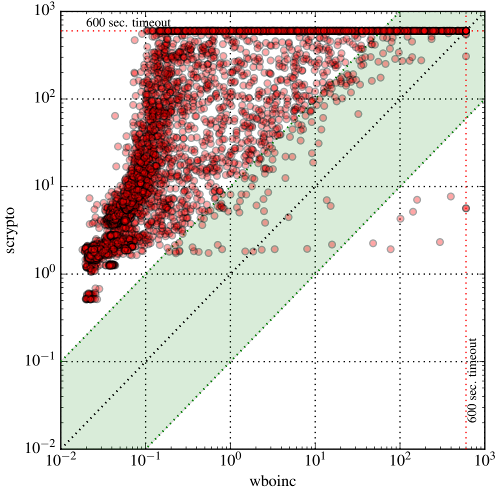


SAT can make the difference – axiom pinpointing



- $\mathcal{EL}^{+}$ medical ontologies [SAT'15]
  - Minimal unsatisfiability (MUSes) & maximal satisfiability (MCSes) & Enumeration

SAT can make the difference – model based diagnosis



- Model-based diagnosis problem instances [IJCAI'15]
  - Maximum satisfiability (MaxSAT)

This tutorial

- Part #1: Modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Goal:** Overview for non-experts

This tutorial

- Part #1: Modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Goal:** Overview for non-experts
- Part #2: Modeling problems for SAT
  - Propositional encodings
  - Modeling examples

This tutorial

- Part #1: Modern SAT solvers
  - **Conflict-Driven Clause Learning** (**CDCL**) SAT solvers
    - **Goal:** Overview for non-experts
- Part #2: Modeling problems for SAT
  - Propositional encodings
  - Modeling examples
- Part #3: Problem solving with SAT oracles
  - Minimal unsatisfiability (MUS)
  - Maximum satisfiability (MaxSAT)
  - ~~Maximal satisfiability~~ (MSS/MCS)
  - ~~Enumeration problems~~
  - ~~Counting problems~~
  - ~~Quantification problems~~
  - ~~Etc.~~

Part I

CDCL SAT Solving

Outline

Basic Definitions

Clause Learning, UIPs & Minimization

Search Restarts & Lazy Data Structures

Why CDCL Works?

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
- 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
- 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,R65]

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

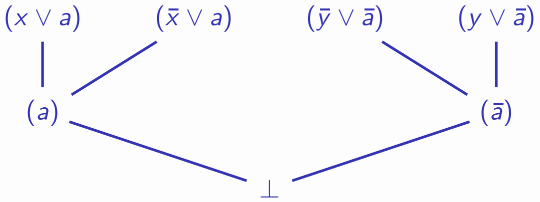
  - Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



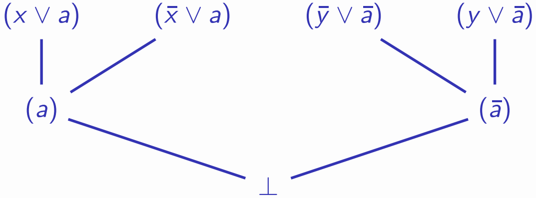
  - Extensively used with (CDCL) SAT solvers

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



  - Extensively used with (CDCL) SAT solvers

- Self-subsuming resolution (with $\alpha' \subseteq \alpha$): [E.g. SP04,EB05]

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

  - (α) subsumes $(\alpha \vee x)$

Unit propagation

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

Unit propagation

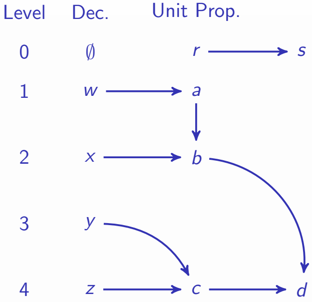
$$\begin{aligned}\mathcal{F} \;=\; & (r) \wedge (\bar{r} \vee s) \wedge \\ & (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ & (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

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

Unit propagation

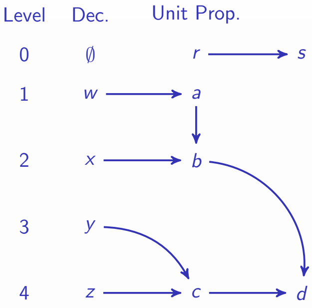
$$\mathcal{F} = (r) \wedge (\bar{r} \vee s) \wedge \\ (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \wedge \\ (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$

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



Unit propagation

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

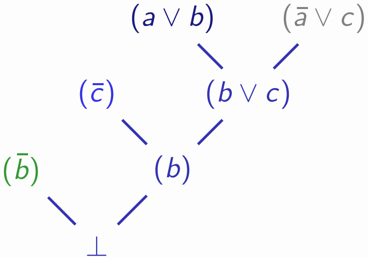


- Decisions / Variable Branchings: $w = 1, x = 1, y = 1, z = 1$

- 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 produces 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:

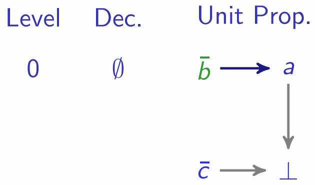


- A modern SAT solver can generate resolution proofs using clauses learned by the solver [ZM03]

Unsatisfiable cores & proof traces

- CNF formula:

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

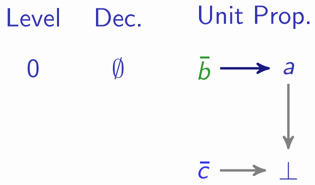


Implication graph with conflict

Unsatisfiable cores & proof traces

- CNF formula:

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

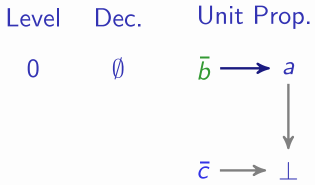


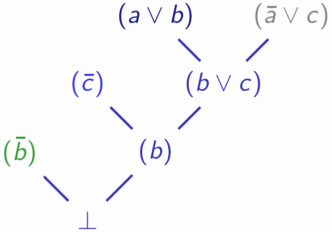
Proof trace $\bot$: $(\bar{a} \vee c)$ $(a \vee b)$ $(\bar{c})$ $(\bar{b})$

Unsatisfiable cores & proof traces

- CNF formula:

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

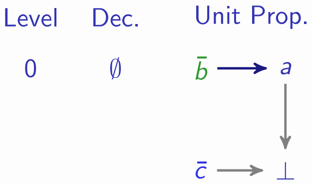


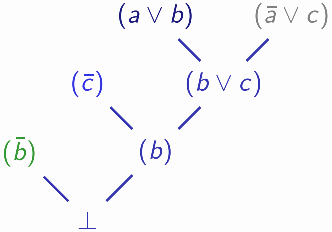


Resolution proof follows structure of conflicts

- CNF formula:

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

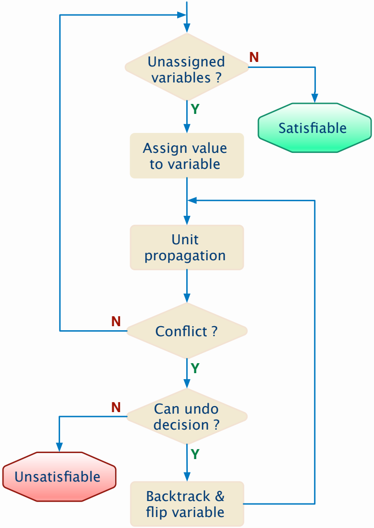




Unsatisfiable subformula (core): $(\bar{c}), (\bar{b}), (\bar{a} \vee c), (a \vee b)$

The DPLL algorithm

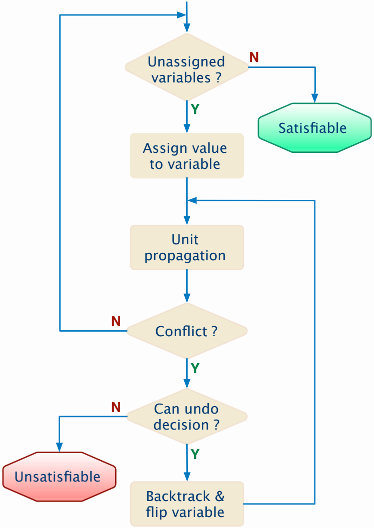
[DL60,DLL62]



- Optional: pure literal rule

The DPLL algorithm

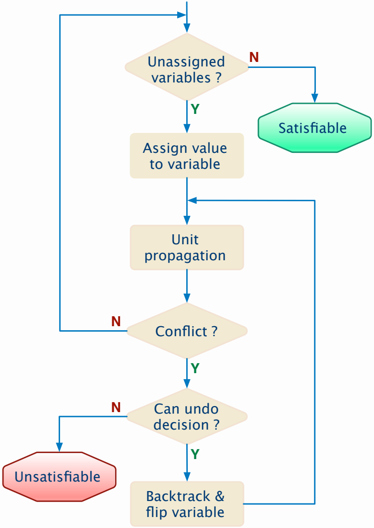
[DL60,DLL62]



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

- Optional: pure literal rule

The DPLL algorithm

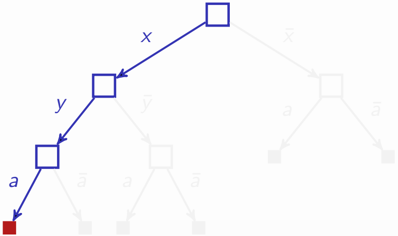


- Optional: pure literal rule

[DL60,DLL62]

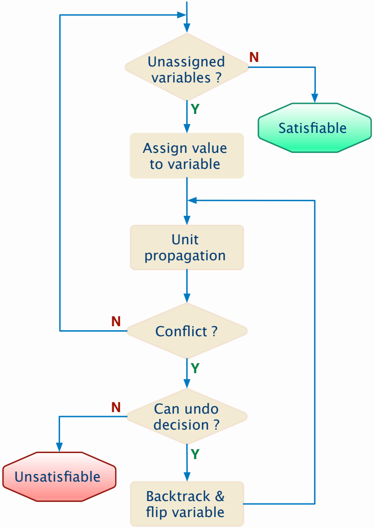
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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



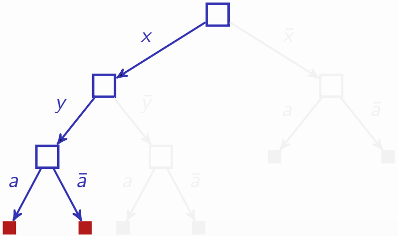
The DPLL algorithm

[DL60,DLL62]



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

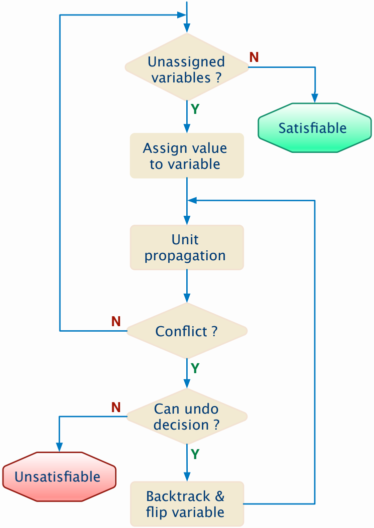
Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	y	
3	$\bar{a}$	$\rightarrow \bar{b} \rightarrow \bot$



- Optional: pure literal rule

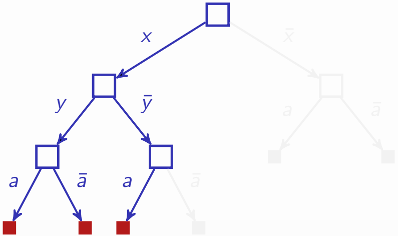
The DPLL algorithm

[DL60,DLL62]



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

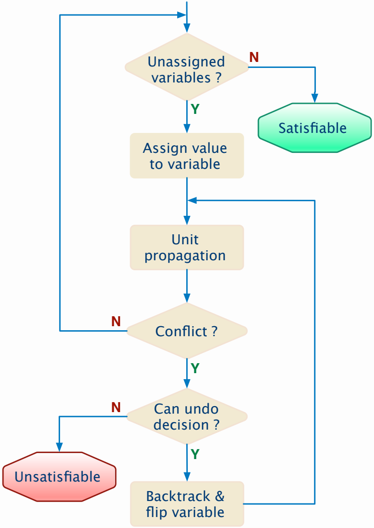
Level	Dec.	Unit Prop.
0	$\emptyset$	
1	x	
2	$\bar{y}$	
3	a	$\to b \to \bot$



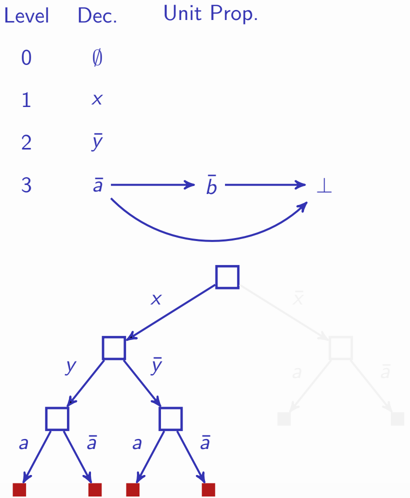
- Optional: pure literal rule

The DPLL algorithm

[DL60,DLL62]



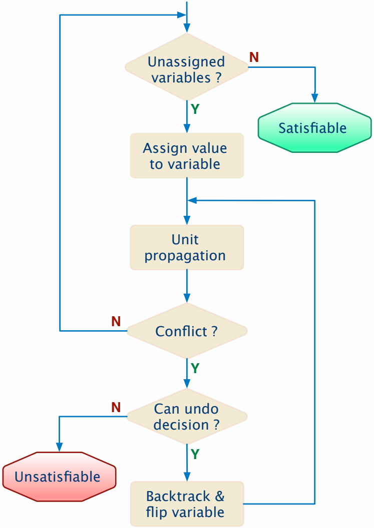
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$



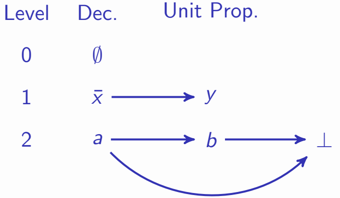
- Optional: pure literal rule

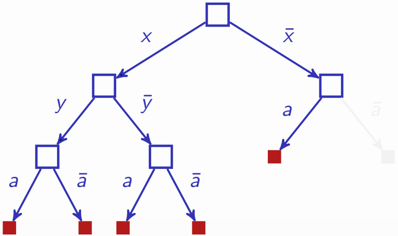
The DPLL algorithm

[DL60,DLL62]



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

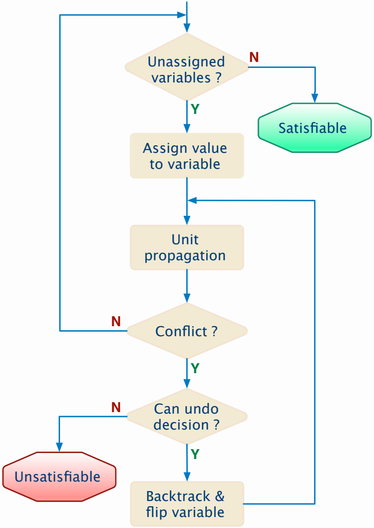




- Optional: pure literal rule

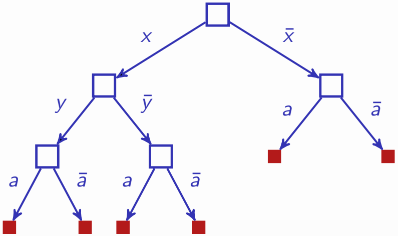
The DPLL algorithm

[DL60,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	$\bar{x}$	$\longrightarrow y$
2	$\bar{a}$	$\longrightarrow \bar{b} \longrightarrow \bot$



- Optional: pure literal rule

How significant is CDCL SAT solving?

- Sample 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. ...

  – **Demo 1**: model checking example (from IBM)

How significant is CDCL SAT solving?

- Sample of solvers:
  1. POSIT: state of the art DPLL SAT solver in 1995
  2. GRASP: first CDCL SAT solver, state of the art 1995$\sim$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. ...

  – **Demo 1**: model checking example (from IBM)

- **Demo 2**: cooperative path finding (CPF)

What is a CDCL SAT solver?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96a,MSS99,BS97,Z97]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
  - Conflict-guided branching
  - ...

What is a CDCL SAT solver?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96a,MSS99,BS97,Z97]
    - Exploit UIPs [MSS96a,SSS12]
    - Minimize learned clauses [SB09,VG09]
    - Opportunistically delete clauses [MSS96a,MSS99,GN02]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
    - Watched literals [MMZZM01]
  - Conflict-guided branching
    - Lightweight branching heuristics [MMZZM01]
    - Phase saving [S00,PD07]
  - ...

Outline

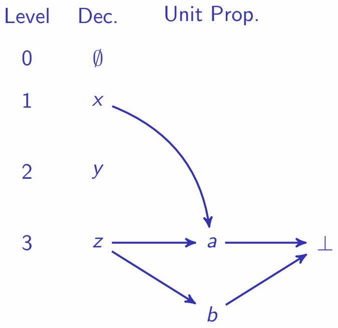
Basic Definitions

Clause Learning, UIPs & Minimization

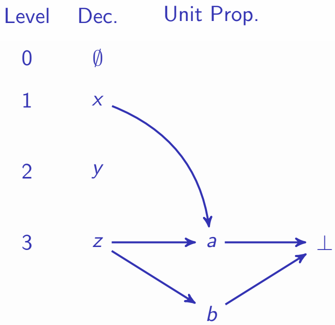
Search Restarts & Lazy Data Structures

Why CDCL Works?

Clause learning



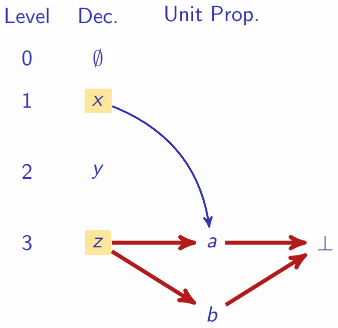
Clause learning



- Analyze conflict

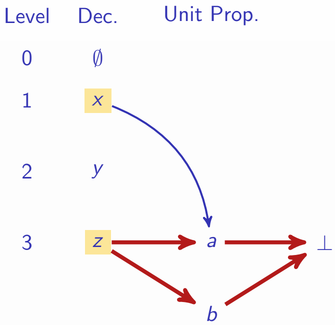
[MSS96a,MSS96b,MSS96c,MSS96d,MSS99]

Clause learning



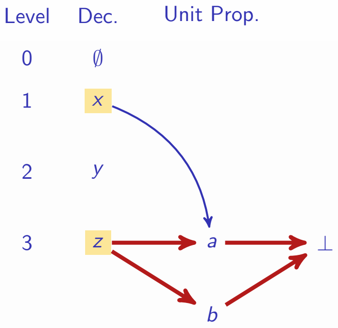
- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current

Clause learning



- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

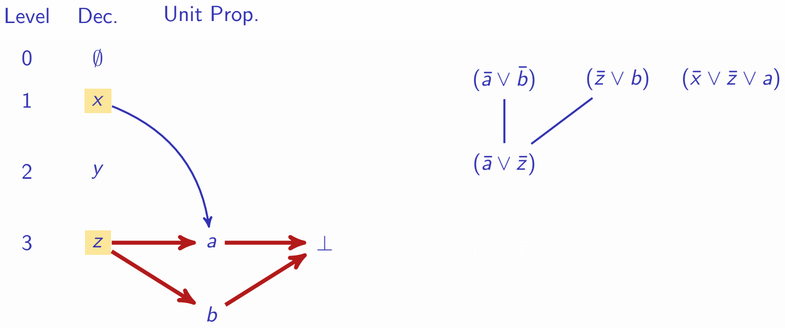
Clause learning



$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

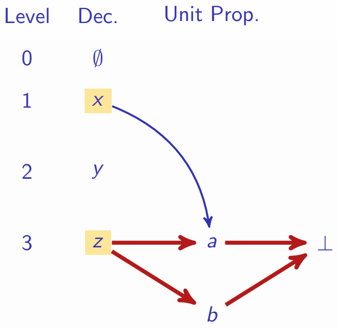
- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

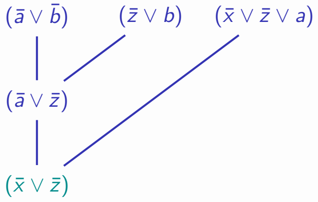
Clause learning



- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,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

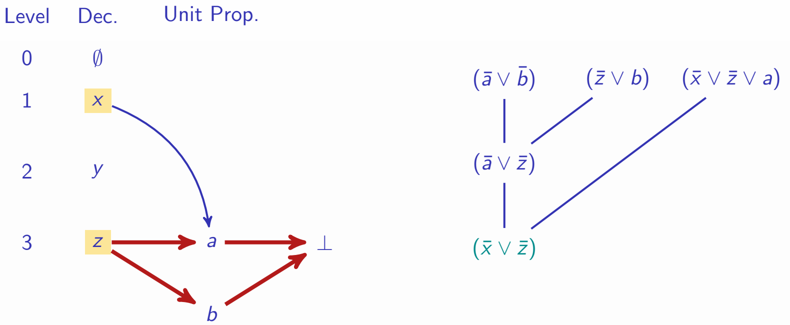
Clause learning



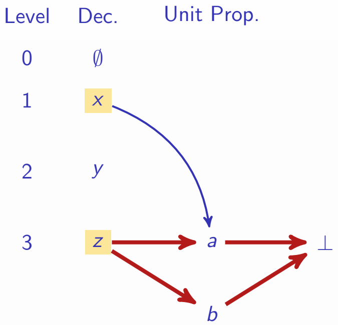


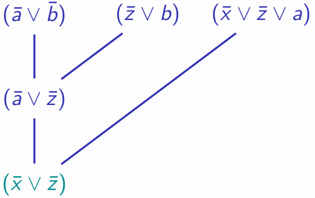
- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \lor \bar{z})$
- Can relate clause learning with resolution

Clause learning



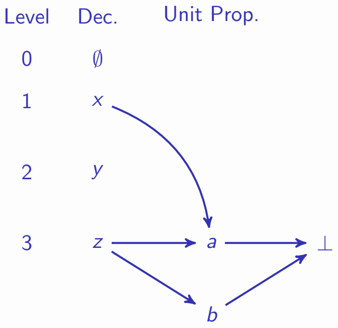
- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations



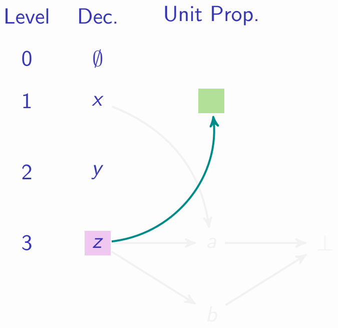


- Analyze conflict [MSS96a,MSS96b,MSS96c,MSS96d,MSS99]
  - Reasons: x and z
    - Decision variable & literals assigned at decision levels less than current
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations
- **Note: GRASP-like clause learning**
  - Other instantiations of clause learning exist

Clause learning – after backtracking

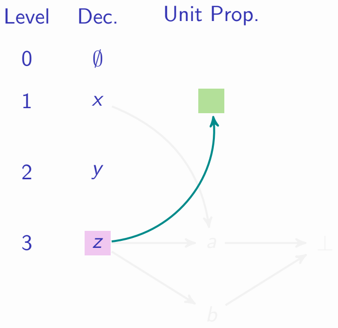


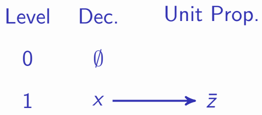
Clause learning – after backtracking



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

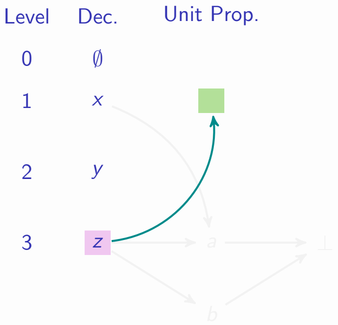
Clause learning – after backtracking

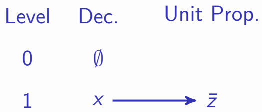




- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

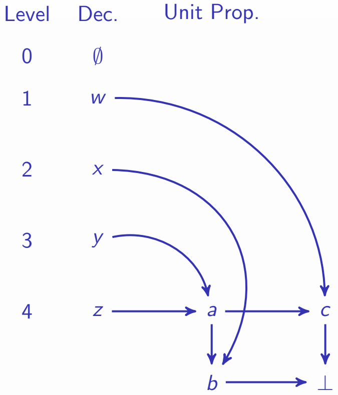
Clause learning – after backtracking



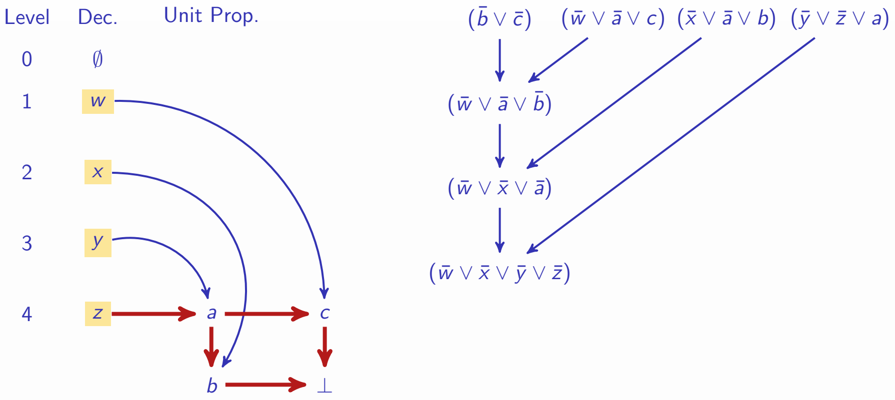


- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are asserting (with exceptions) [MSS96a,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [ZMMM01]

Unique implication points (UIPs)

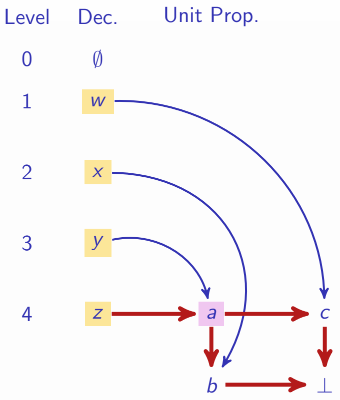


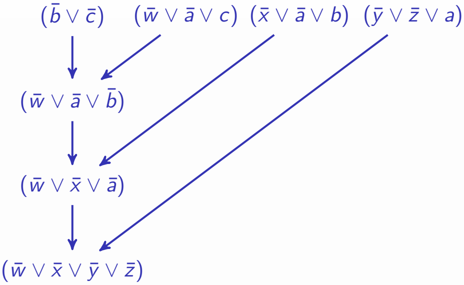
Unique implication points (UIPs)



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

Unique implication points (UIPs)

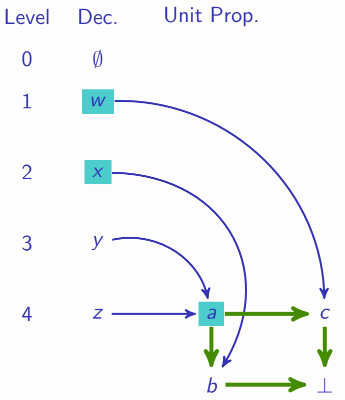


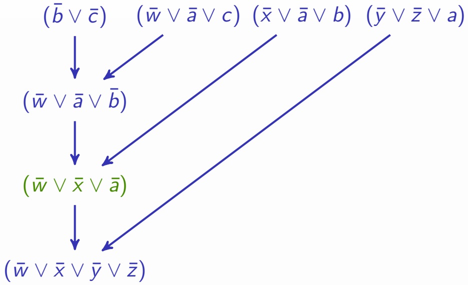


- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP
  - Dominator in DAG for decision level 4

[MSS96a,MSS99]

Unique implication points (UIPs)

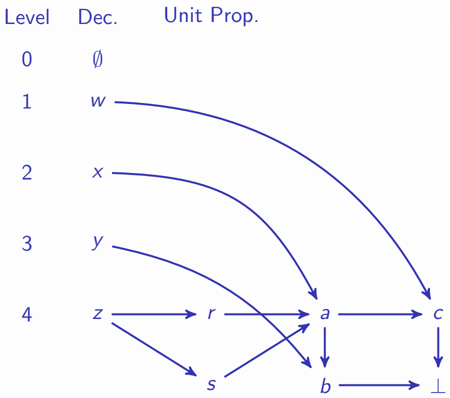




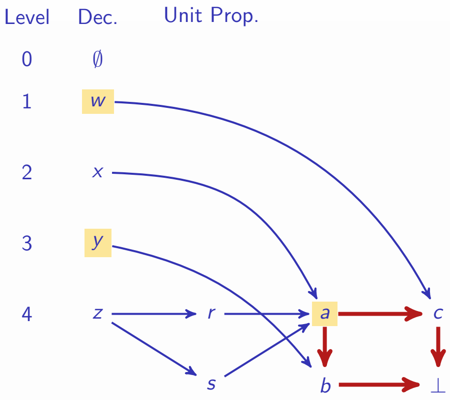
- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
  - Dominator in DAG for level 4
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

[MSS96a,MSS99]

Multiple UIPs

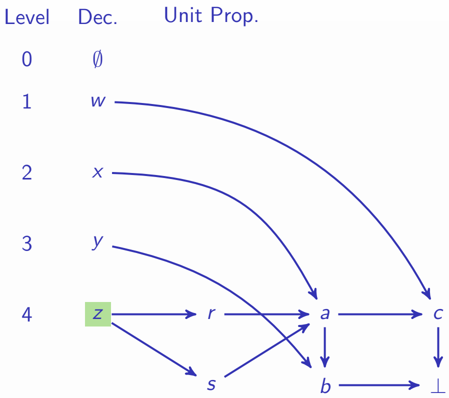


Multiple UIPs



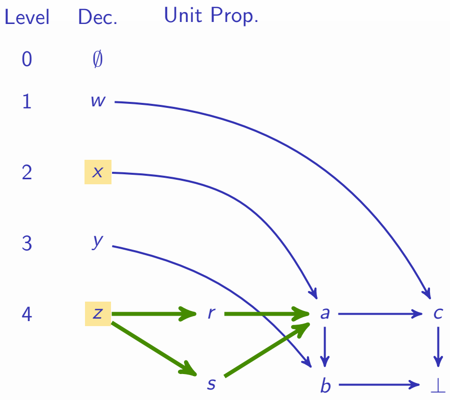
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$

Multiple UIPs



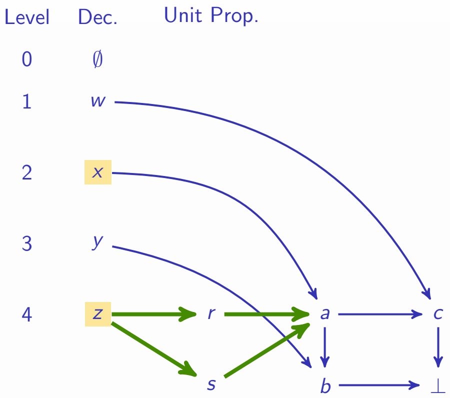
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP

Multiple UIPs



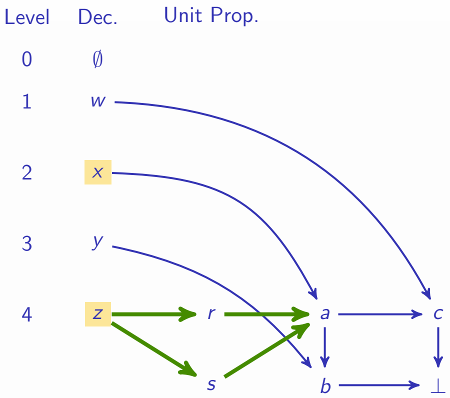
- First UIP:
  - Learn clause $(\bar{w} \lor \bar{y} \lor \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \lor \bar{z} \lor a)$
  - Clause is not asserting

Multiple UIPs



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

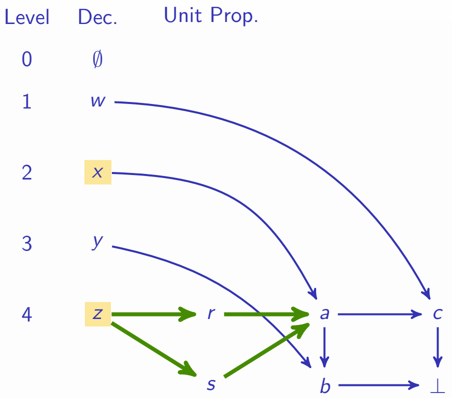
Multiple UIPs



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

- Multiple UIPs proposed in GRASP [MSS96a,MSS99]
  - First UIP learning proposed in Chaff [ZMMM01]
- Not used in recent state of the art CDCL SAT solvers

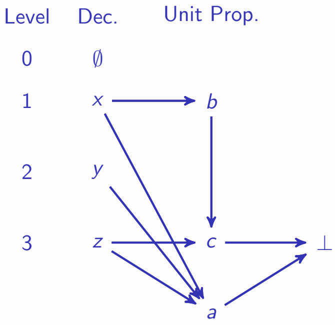
Multiple UIPs



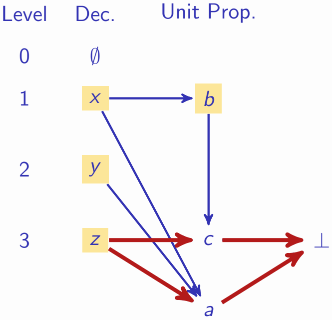
- First UIP:
  - Learn clause $(\bar{w} \vee \bar{y} \vee \bar{a})$
- But there can be more than 1 UIP
- Second UIP:
  - Learn clause $(\bar{x} \vee \bar{z} \vee a)$
  - Clause is not asserting
- In practice smaller clauses more effective
  - Compare with $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

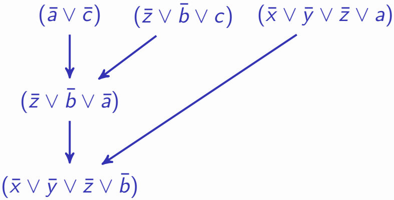
- Multiple UIPs proposed in GRASP [MSS96a,MSS99]
  - First UIP learning proposed in Chaff [ZMMM01]
- Not used in recent state of the art CDCL SAT solvers
- Recent results show it can be beneficial on some instances [SSS12]

Clause minimization I



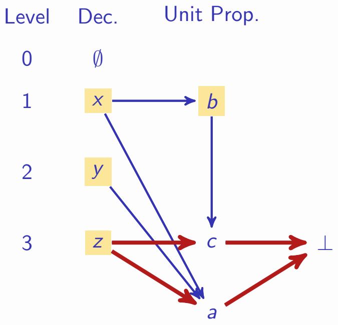
Clause minimization I

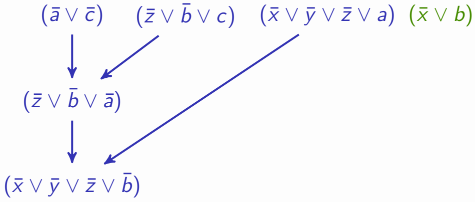




- Learn clause $(\bar{x} \lor \bar{y} \lor \bar{z} \lor \bar{b})$

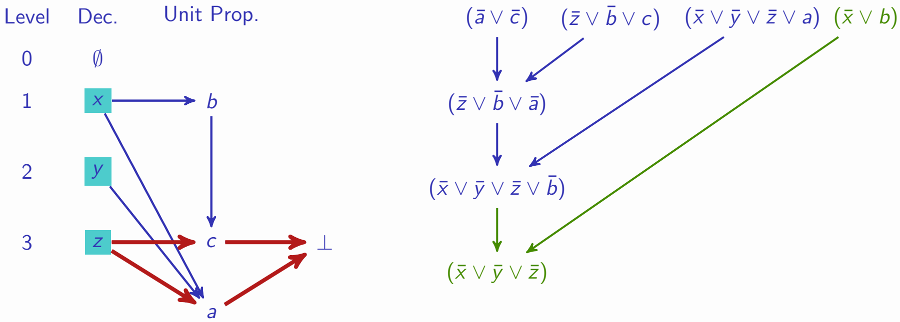
Clause minimization I





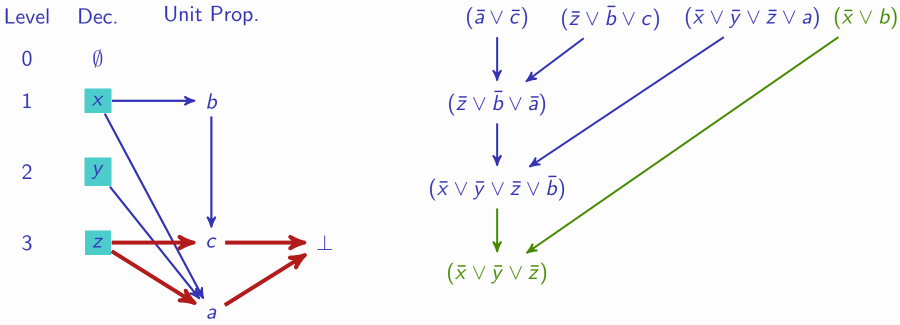
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

Clause minimization I



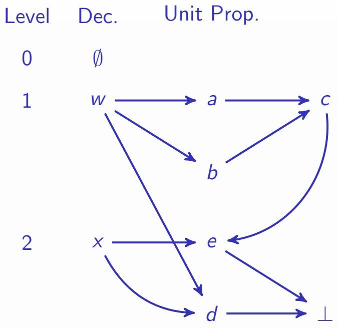
- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

Clause minimization I

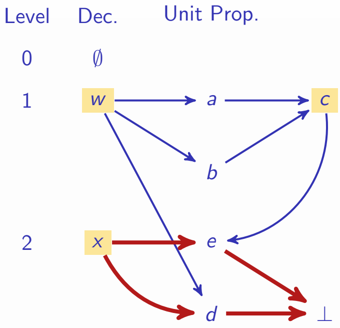


- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization) [SB09]
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

Clause minimization II

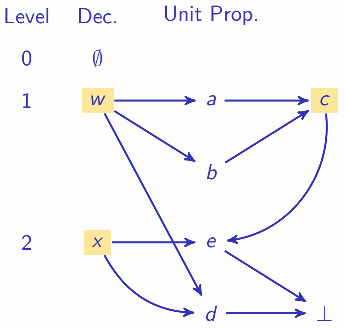


Clause minimization II



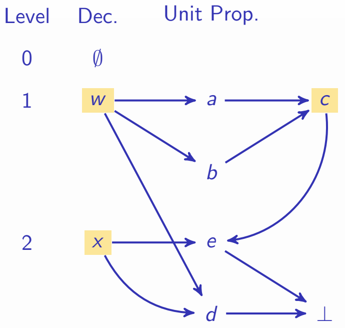
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$

Clause minimization II



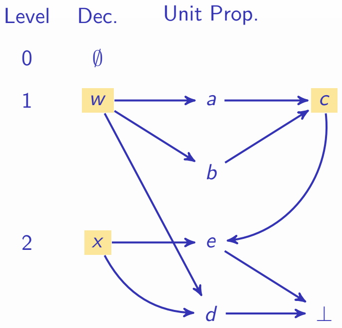
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{c})$
- Cannot apply self-subsuming resolution
  - Resolving with reason of c yields $(\bar{w} \vee \bar{x} \vee \bar{a} \vee \bar{b})$

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

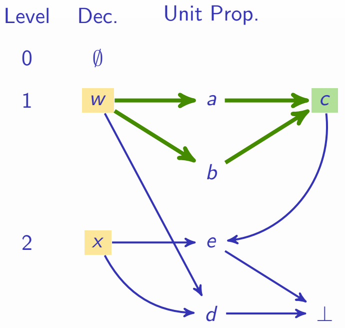
Clause minimization II



- ~~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]

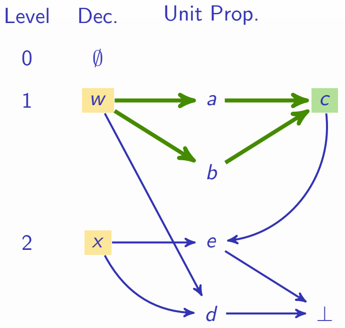
Clause minimization II



- ~~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

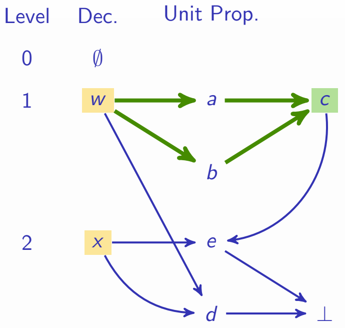
Clause minimization II



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

- 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

Outline

Basic Definitions

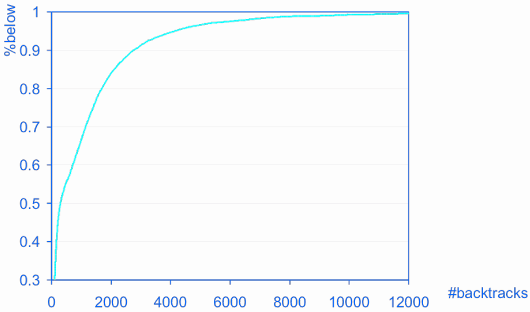
Clause Learning, UIPs & Minimization

Search Restarts & Lazy Data Structures

Why CDCL Works?

Search restarts I

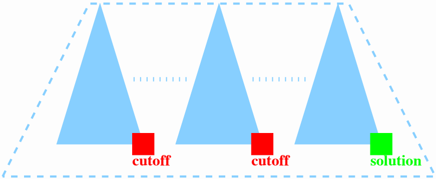
- Heavy-tail behavior: [GSK98]



  - 10000 runs, branching randomization on satisfiable industrial instance
    - Use rapid randomized restarts (search restarts)

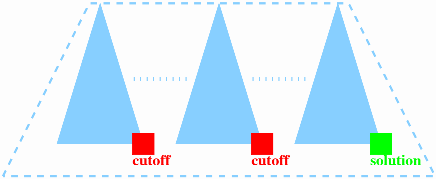
Search restarts II

- Restart search after a number of conflicts



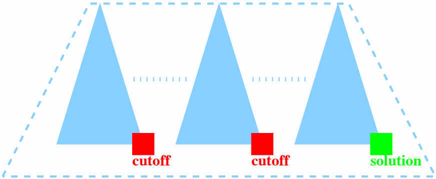
Search restarts II

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



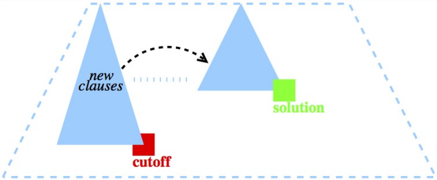
Search restarts II

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist
- Works for SAT & UNSAT instances. **Why?**
  - Not explained by heavy-tailed behavior



Search restarts II

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist
- Works for SAT & UNSAT instances. **Why?**
  - Not explained by heavy-tailed behavior
  - But there exist proof complexity arguments
- Learned clauses effective after restart(s)



Data structures basics

- Each literal l should access clauses containing l
  - Why?

Data structures basics

- Each literal l should access clauses containing l
  - Why? Unit propagation

Data structures basics

- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause

Data structures basics

- Each literal l should access clauses containing 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

- Each literal l should access clauses containing 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

- Each literal l should access clauses containing 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

- Each literal l should access clauses containing 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

- Each literal l should access clauses containing 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

- Each literal l should access clauses containing 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** [MMZZM01]

Data structures basics

- Each literal l should access clauses containing 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,

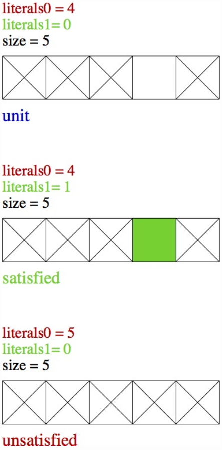
    Unit propagation slow-down worse than linear as clauses are learned **!**

- Clause learning to be effective requires a more efficient representation: **Watched Literals** [MMZZM01]
  - Watched literals are one example of lazy data structures
    - But there are others

Watched literals

[MMZZM01]

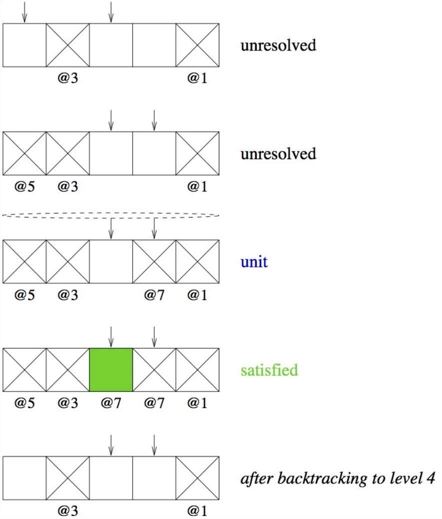
- Important states of a clause



Watched literals

[MMZZM01]

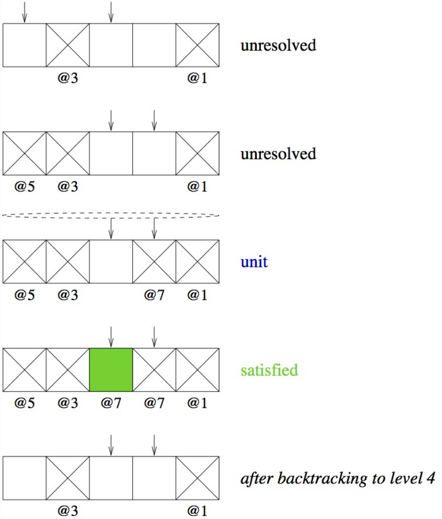
- Important states of a clause
- Associate **2** references with each clause



Watched literals

[MMZZM01]

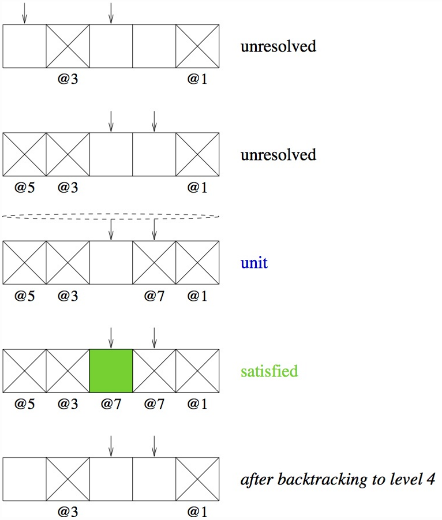
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



Watched literals

[MMZZM01]

- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals
- References **unchanged** when backtracking



Additional key techniques

- Lightweight branching [MMZZM01]
  - 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

- Lightweight branching [MMZZM01]
  - 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 [E.g. MSS96a,MSS99]
  - Delete less used clauses [E.g. GN02,ES03]

Additional key techniques

- Lightweight branching [MMZZM01]
  - 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 [E.g. MSS96a,MSS99]
  - Delete less used clauses [E.g. GN02,ES03]
- Other effective techniques:
  - Phase saving [S00,PD07]
  - Luby restarts [H07]
  - Literal blocks distance [AS09]
  - Preprocessing/inprocessing [E.g. JHB12,HJLSB15]

Outline

Basic Definitions

Clause Learning, UIPs & Minimization

Search Restarts & Lazy Data Structures

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 [MSS94a,MSS94b]

- There are also proof complexity arguments

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 [MSS94a,MSS94b]
- 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 [S13]
- There are also proof complexity arguments

Part II

Problem Modeling for SAT

Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

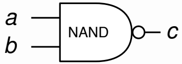
Linear Constraints

Encoding CSPs

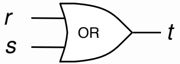
Modeling Examples & Exercises

- 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 [T68,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

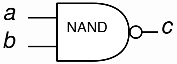
$$\mathcal{F}_c = (a \vee c) \wedge (b \vee c) \wedge (\bar{a} \vee \bar{b} \vee \bar{c})$$



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



Representing Boolean formulas / circuits II



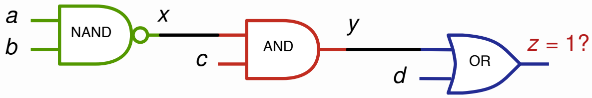
c \ ab	00	01	11	10
0	0	0	1	0
1	1	1	0	1

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

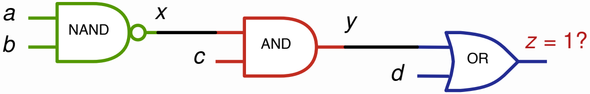
- CNF formula for the circuit is the conjunction of the CNF formula for each gate
  - Can specify objectives with additional clauses



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

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

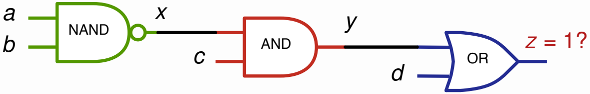


$$
\begin{aligned}
\mathcal{F} \;=\;& (a \vee x) \wedge (b \vee x) \wedge (\bar{a} \vee \bar{b} \vee \bar{x}) \wedge \\
& (x \vee \bar{y}) \wedge (c \vee \bar{y}) \wedge (\bar{x} \vee \bar{c} \vee y) \wedge \\
& (\bar{y} \vee z) \wedge (\bar{d} \vee z) \wedge (y \vee d \vee \bar{z}) \wedge (z)
\end{aligned}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

- 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



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

- 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.

Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples & Exercises

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?

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

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?

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

  - Hard constraints: φ
  - Soft constraints: (x_j), each with cost c_j

Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples & Exercises

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 [S05]
    - Clauses: $\mathcal{O}(n)$; Auxiliary variables: $\mathcal{O}(n)$
  - Bitwise encoding [P07,FP01]
    - 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 $a + b + c + d \leq 1$?

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

  - 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 $a + b + c + d \leq 1$?

$$\begin{array}{lcl} a \rightarrow \bar{b} \wedge \bar{c} \wedge \bar{d} & \Longrightarrow & (\bar{a} \vee \bar{b}) \wedge (\bar{a} \vee \bar{c}) \wedge (\bar{a} \vee \bar{d}) \\ b \rightarrow \bar{c} \wedge \bar{d} \wedge \bar{a} & \Longrightarrow & (\bar{b} \vee \bar{c}) \wedge (\bar{b} \vee \bar{d}) \wedge (\bar{b} \vee \bar{a}) \\ c \rightarrow \bar{d} \wedge \bar{a} \wedge \bar{b} & \Longrightarrow & (\bar{c} \vee \bar{d}) \wedge (\bar{c} \vee \bar{a}) \wedge (\bar{c} \vee \bar{b}) \\ d \rightarrow \bar{a} \wedge \bar{b} \wedge \bar{c} & \Longrightarrow & (\bar{d} \vee \bar{a}) \wedge (\bar{d} \vee \bar{b}) \wedge (\bar{d} \vee \bar{c}) \end{array}$$

  - 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:

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

  - 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 [W98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - Generalized pairwise
    - Clauses: $\mathcal{O}(2^n)$; no auxiliary variables
  - Sequential counters [S05]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - BDDs [ES06]
    - Clauses/Variables: $\mathcal{O}(n\,k)$
  - Sorting networks [ES06]
    - Clauses/Variables: $\mathcal{O}(n \log^2 n)$
  - Cardinality Networks: [ANORC09,ANORC11a]
    - Clauses/Variables: $\mathcal{O}(n \log^2 k)$
  - Pairwise Cardinality Networks: [CZI10]
  - ...

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

$$\begin{array}{lcl} a \wedge b \rightarrow \bar{c} & \Longrightarrow & (\bar{a} \vee \bar{b} \vee \bar{c}) \\ a \wedge b \rightarrow \bar{d} & \Longrightarrow & (\bar{a} \vee \bar{b} \vee \bar{d}) \\ a \wedge c \rightarrow \bar{d} & \Longrightarrow & (\bar{a} \vee \bar{c} \vee \bar{d}) \\ b \wedge c \rightarrow \bar{d} & \Longrightarrow & (\bar{b} \vee \bar{c} \vee \bar{d}) \end{array}$$

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

$$
\begin{array}{lcl}
a \wedge b \rightarrow \bar{c} & \Longrightarrow & (\bar{a} \vee \bar{b} \vee \bar{c}) \\
a \wedge b \rightarrow \bar{d} & \Longrightarrow & (\bar{a} \vee \bar{b} \vee \bar{d}) \\
a \wedge c \rightarrow \bar{d} & \Longrightarrow & (\bar{a} \vee \bar{c} \vee \bar{d}) \\
b \wedge c \rightarrow \bar{d} & \Longrightarrow & (\bar{b} \vee \bar{c} \vee \bar{d})
\end{array}
$$

  - 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: $\frac{n(n-1)}{2}$

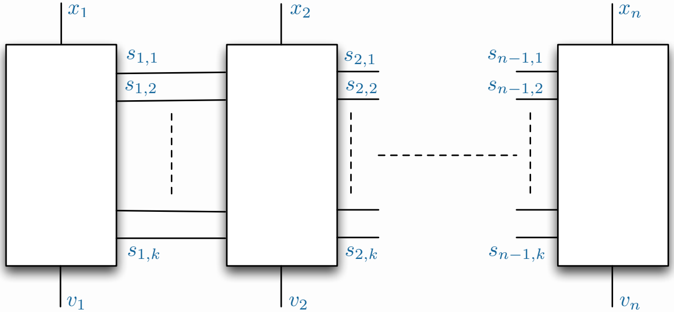
Another example

- 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: $1 < i < n \;,\; 1 < j \leq k$

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

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

Pseudo-Boolean constraints

- General form: $\sum_{j=1}^{n} a_j\, x_j \leq b$
  - Operational encoding [W98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses

Pseudo-Boolean constraints

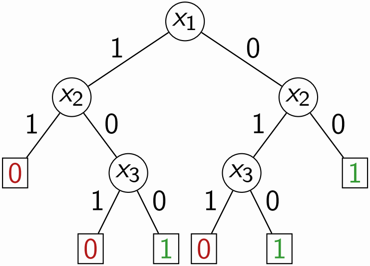
- General form: $\sum_{j=1}^{n} a_j x_j \leq b$
  - Operational encoding [W98]
    - Clauses/Variables: $\mathcal{O}(n)$
    - Does not guarantee arc-consistency
  - BDDs [ES06]
    - Worst-case exponential number of clauses
  - Polynomial watchdog encoding [BBR09]
    - Let $\nu(n) = \log(n) \log(a_{max})$
    - 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 [W98]
    - 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 [ANORC11b]
    - Clauses & aux variables: $\mathcal{O}(n^3\log(a_{max}))$
  - ...

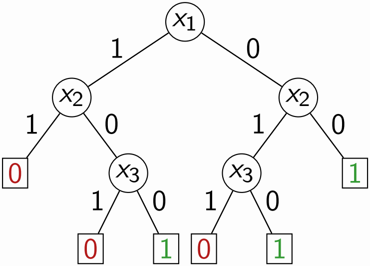
Encoding PB constraints with BDDs I

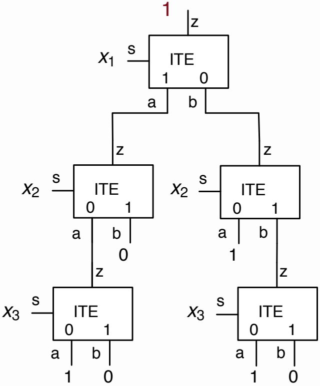
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD



Encoding PB constraints with BDDs I

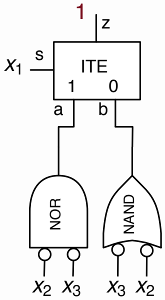
- Encode $3x_1 + 3x_2 + x_3 \leq 3$
- Construct BDD
  - E.g. analyze variables by decreasing coefficients
- Extract ITE-based circuit from BDD





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 subset-sum constraint
      (special case of 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 subset-sum constraint
      (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

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 subset-sum constraint (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- 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 subset-sum constraint (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- 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 subset-sum constraint (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- 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 subset-sum constraint (special case of a knapsack constraint)
    - Cannot find all consequences in polynomial time [S03,FS02,T03]

- 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

Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

Linear Constraints

Encoding CSPs

Modeling Examples & Exercises

CSP constraints

- Many possible encodings:
  - Direct encoding [dK89,GJ96,W00]
  - Log encoding [W00]
  - Support encoding [K90,G02]
  - Log-Support encoding [G07]
  - 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 $\sum_{k=1}^{m_i} x_{i,k} \geq 1$ [W00]
- 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
  - ...

Outline

Recap Clausification of Boolean Formulas

Hard and Soft Constraints

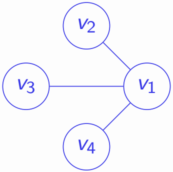
Linear Constraints

Encoding CSPs

Modeling Examples & Exercises

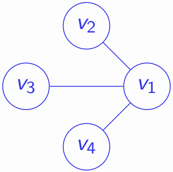
Minimum vertex cover

- 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

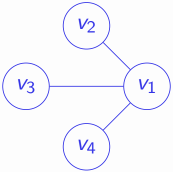
- The problem:
  - Graph $G = (V, E)$
  - Vertex cover $U \subseteq V$
    - For each $(v_i, v_j) \in E$, either $v_i \in U$ or $v_j \in U$
  - Minimum vertex cover: vertex cover U of minimum size



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

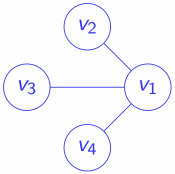
Min vertex cover: $\{v_1\}$

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

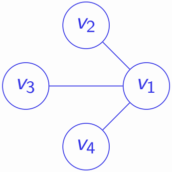
- Modeling with Pseudo-Boolean Optimization (PBO):
  - Variables: x_i for each $v_i \in V$, with $x_i = 1$ iff $v_i \in U$
  - Clauses: $(x_i \vee x_j)$ for each $(v_i, v_j) \in E$
  - Objective function: minimize number of true x_i variables
    - I.e. minimize vertices included in U



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

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:

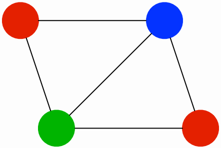
$$
\begin{aligned}
\varphi_S &= \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_4)\} \\
\varphi_H &= \{(x_1 \vee x_2), (x_1 \vee x_3), (x_1 \vee x_4)\}
\end{aligned}
$$

Graph coloring

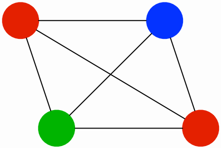
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?

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?



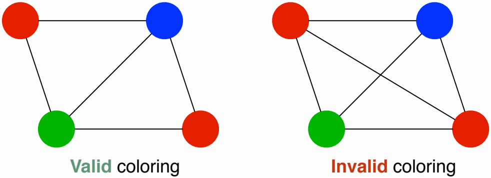
Valid coloring



Invalid coloring

Graph coloring

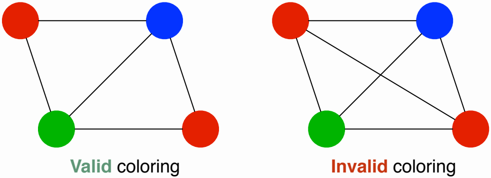
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- How to model color assignments to vertices?

Graph coloring

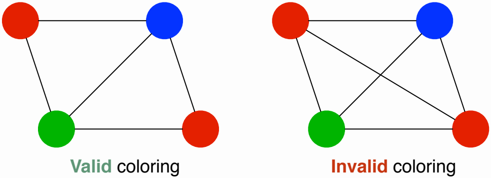
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- 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

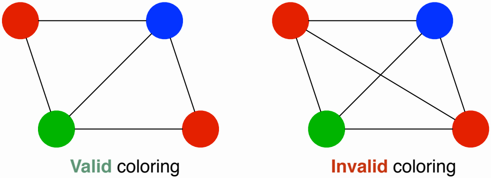
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- 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

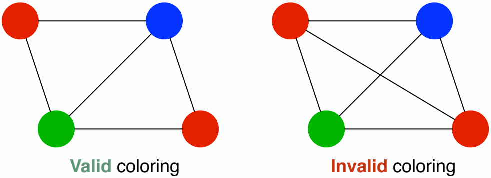
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- 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

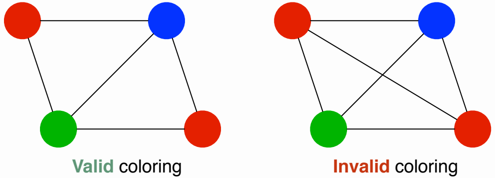
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- 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

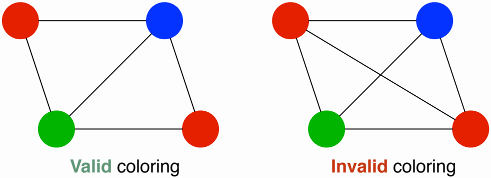
- Given undirected graph $G = (V, E)$ and k colors:
  - Can we assign colors to vertices of G s.t. any pair of adjacent vertices are assigned different colors?



- 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:

Q				
			Q	
	Q			
				Q
		Q		

The N-Queens problem II

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

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

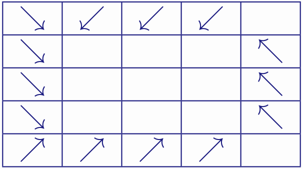
- Each column must have exactly one queen:

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

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

The N-Queens problem III

- Each diagonal can have at most one queen:



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

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

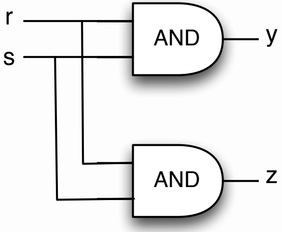
$$j = 1, \quad 1 \leq i < N, \quad \sum_{k=0}^{N-i} x_{i+k\, j+k} \leq 1$$

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

Design debugging

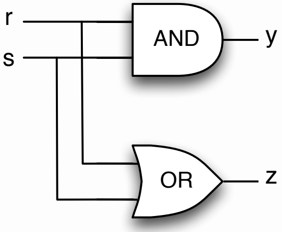
[SMVLS'07]

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

[MBCV'06,TSJL'07,AL'08,ALMS'09,ALBL'10]

- 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

[MBCV'06,TSJL'07,AL'08,ALMS'09,ALBL'10]

- 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:

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

Exercise: knapsack

- 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

Exercise: knapsack

- 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

Exercise: knapsack

- 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?

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

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

Exercise: solving Sudoku I

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

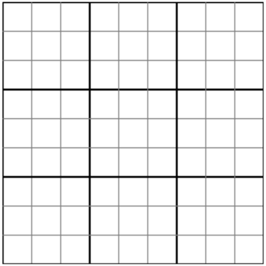
Exercise: solving Sudoku II

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Exercise: solving Sudoku II

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

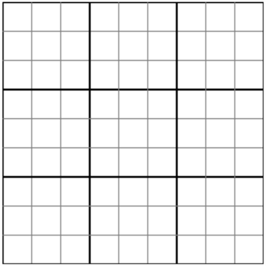
- How to solve Sudoku with SAT?



- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i, j \in \{1, \ldots, 9\}$

- Constraints:

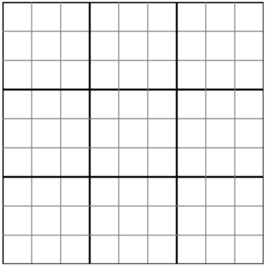
Solving Sudoku – with constraints



- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i, j \in \{1, \ldots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{i,1}, \ldots, v_{i,9})$

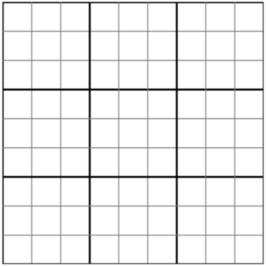
Solving Sudoku – with constraints



- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i, j \in \{1, \ldots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{i,1}, \ldots, v_{i,9})$
  - Each value used exactly once in each column:
    - For $j \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{1,j}, \ldots, v_{9,j})$

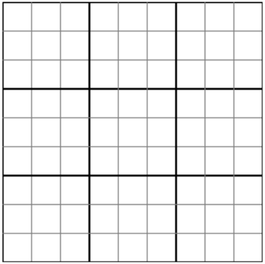
Solving Sudoku – with constraints



- Modeling the problem with integer variables:
  - Rows: $i = 1, \ldots, 9$
  - Columns: $j = 1, \ldots, 9$
  - Variables: $v_{i,j} \in \{1, 2, \ldots, 9\}$, $i, j \in \{1, \ldots, 9\}$

- Constraints:
  - Each value used exactly once in each row:
    - For $i \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{i,1}, \ldots, v_{i,9})$
  - Each value used exactly once in each column:
    - For $j \in \{1, \ldots, 9\}$: $\text{alldifferent}(v_{1,j}, \ldots, v_{9,j})$
  - Each value used exactly once in each 3×3 sub-grid:
    - For $i, j \in \{0, 1, 2\}$:
      $\text{alldifferent}(v_{3i+1,3j+1}, v_{3i+1,3j+2}, v_{3i+1,3j+3}, v_{3i+2,3j+1}, \ldots, v_{3i+3,3j+1}, \ldots)$

Solving Sudoku – propositional logic – variables



- Modeling with propositional variables:
  - Rows: $i = 1, \dots, 9$
  - Columns: $j = 1, \dots, 9$
  - Variables: $v_{i,j,k} \in \{0, 1\}$, $i, j, k \in \{1, \dots, 9\}$

Solving Sudoku – propositional logic – constraints

- Value in each cell is valid:
  - For $i, j \in \{1, \ldots, 9\}$:

$$\sum_{k=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each row:
  - For $i \in \{1, \ldots, 9\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{j=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each column:
  - For $j \in \{1, \ldots, 9\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{i=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each 3×3 sub-grid:
  - For $i, j \in \{0, 1, 2\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{r=1}^{3} \sum_{s=1}^{3} v_{3i+r,3j+s,k} = 1$$

Solving Sudoku – propositional logic – constraints

- Value in each cell is valid:
  - For $i, j \in \{1, \ldots, 9\}$:

$$\sum_{k=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each row:
  - For $i \in \{1, \ldots, 9\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{j=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each column:
  - For $j \in \{1, \ldots, 9\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{i=1}^{9} v_{i,j,k} = 1$$

- Each value used exactly once in each 3×3 sub-grid:
  - For $i, j \in \{0, 1, 2\}$, $k \in \{1, \ldots, 9\}$:

$$\sum_{r=1}^{3} \sum_{s=1}^{3} v_{3i+r,3j+s,k} = 1$$

- **Q**: how to encode Equals1 constraints?

Constraints for fixed cells

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Constraints for fixed cells

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Integer variables:

$$v_{1,1} = 5, v_{1,2} = 3, v_{1,5} = 7, v_{2,1} = 6, v_{2,4} = 1, v_{2,5} = 9$$
$$v_{2,6} = 5, v_{3,2} = 9, v_{3,3} = 8, v_{3,8} = 6, v_{4,1} = 8, v_{4,5} = 6, \ldots$$

Constraints for fixed cells

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Integer variables:

$$v_{1,1} = 5, v_{1,2} = 3, v_{1,5} = 7, v_{2,1} = 6, v_{2,4} = 1, v_{2,5} = 9$$
$$v_{2,6} = 5, v_{3,2} = 9, v_{3,3} = 8, v_{3,8} = 6, v_{4,1} = 8, v_{4,5} = 6, \ldots$$

- Propositional variables:

$$v_{1,1,5} = 1, v_{1,2,3} = 1, v_{1,5,7} = 1, v_{2,1,6} = 1, v_{2,4,1} = 1, v_{2,5,9} = 1$$
$$v_{2,6,5} = 1, v_{3,2,9} = 1, v_{3,3,8} = 1, v_{3,8,6} = 1, v_{4,1,8} = 1, v_{4,5,6} = 1, \ldots$$

Part III

Problem Solving with 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:
    - Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$
    - Consider $\mathcal{F} \wedge (x_i)$. Call NP oracle. If answer is **yes**, then add (x_i) to $\mathcal{F}$. If answer is **no**, then add $(\neg x_i)$ to $\mathcal{F}$

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:
    - Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$
    - Consider $\mathcal{F} \wedge (x_i)$. Call NP oracle. If answer is **yes**, then add (x_i) to $\mathcal{F}$. If answer is **no**, then add $(\neg x_i)$ to $\mathcal{F}$
  - Algorithm needs $|\text{var}(\mathcal{F})|$ 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:
    - Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$
    - Consider $\mathcal{F} \wedge (x_i)$. Call NP oracle. If answer is **yes**, then add (x_i) to $\mathcal{F}$. If answer is **no**, then add $(\neg x_i)$ to $\mathcal{F}$
  - Algorithm needs $|\text{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless P = 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:
    - Analyze each variable $x_i \in \{x_1, \ldots, x_n\} = \text{var}(\mathcal{F})$
    - Consider $\mathcal{F} \wedge (x_i)$. Call NP oracle. If answer is **yes**, then add (x_i) to $\mathcal{F}$. If answer is **no**, then add $(\neg x_i)$ to $\mathcal{F}$
  - Algorithm needs $|\text{var}(\mathcal{F})|$ calls to an NP oracle
  - **Note:** Cannot solve FSAT with logarithmic number of NP oracle calls, unless $\text{P} = \text{NP}$ [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

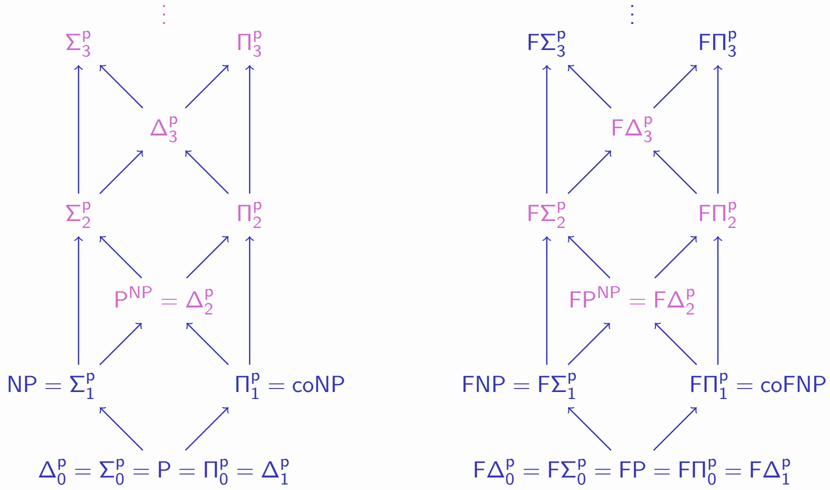
Beyond decision problems

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

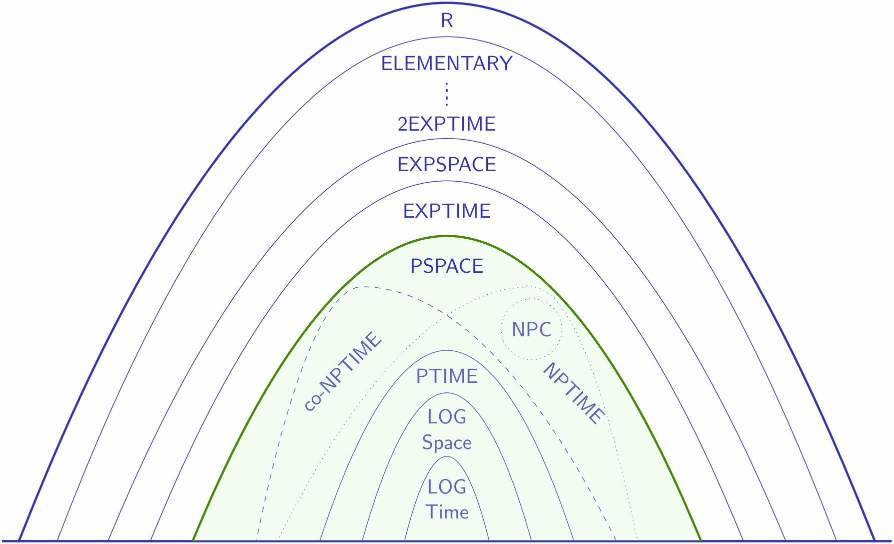
Beyond decision problems

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

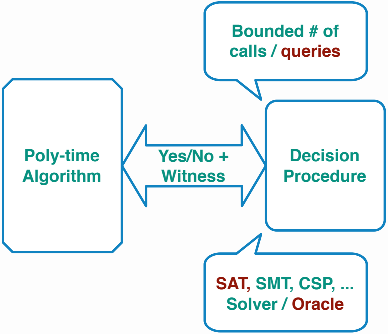
... and beyond NP – decision and function problems



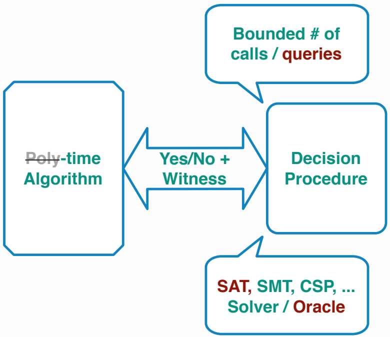
... and beyond NP – our current range



Oracle-based problem solving – ideal scenario



Oracle-based problem solving – in some settings



Many problems to solve – within FP^{NP}

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

Many problems to solve – within FP^{NP}

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

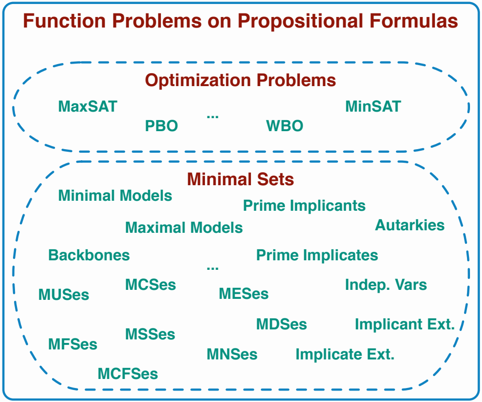
Function Problems on Propositional Formulas

MaxSAT **PBO** **...** **WBO** **MinSAT**

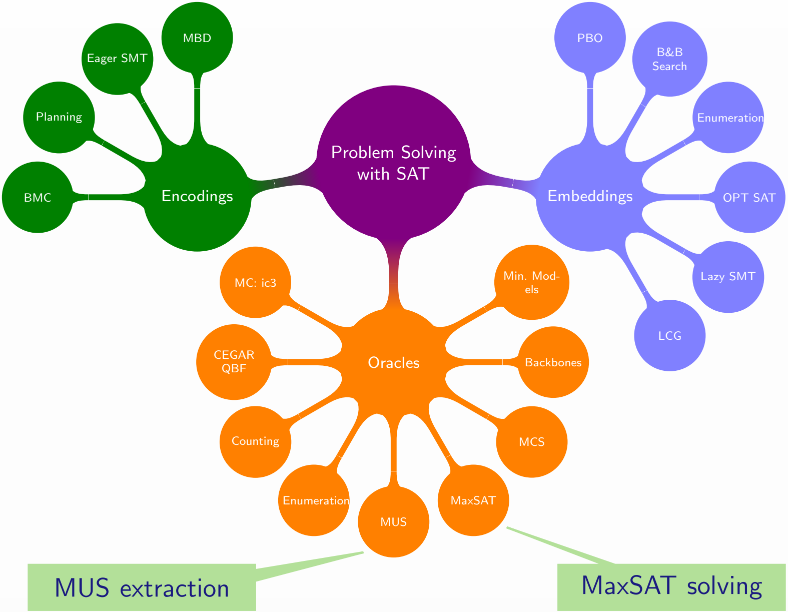
Minimal Models **Prime Implicants** **Maximal Models** **Autarkies** **Backbones** **...** **Prime Implicates** **MUSes** **MCSes** **MESes** **Indep. Vars** **MDSes** **Implicant Ext.** **MFSes** **MSSes** **MNSes** **Implicate Ext.** **MCFSes**

Many problems to solve – within FP^{NP}

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



Selection of topics



Outline

Minimal Unsatisfiability

Maximum Satisfiability

Additional Exercises

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?

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 $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

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

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ ($\vDash \bot$), $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

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

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ $(\vDash \perp)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \perp$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \perp$

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

- Given $\mathcal{F}$ $(\vDash \perp)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \perp$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \perp$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ 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 $\mathcal{F}$ $(\vDash \perp)$, $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \perp$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \perp$

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

- Given $\mathcal{F}$ $(\vDash \perp)$, $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \perp$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \perp$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ 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 $\mathcal{F}$ ($\models \perp$), $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \models \perp$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \perp$

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

- Given $\mathcal{F}$ ($\models \perp$), $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \perp$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \models \perp$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ 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 [R87,...]

Unsatisfiable formulas – MUSes & MCSes

- Given $\mathcal{F}$ ($\vDash \bot$), $\mathcal{M} \subseteq \mathcal{F}$ is a Minimal Unsatisfiable Subset (MUS) iff $\mathcal{M} \vDash \bot$ and $\forall_{\mathcal{M}' \subsetneq \mathcal{M}}, \mathcal{M}' \nvDash \bot$

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

- Given $\mathcal{F}$ ($\vDash \bot$), $\mathcal{C} \subseteq \mathcal{F}$ is a Minimal Correction Subset (MCS) iff $\mathcal{F} \setminus \mathcal{C} \nvDash \bot$ and $\forall_{\mathcal{C}' \subsetneq \mathcal{C}}, \mathcal{F} \setminus \mathcal{C}' \vDash \bot$. $\mathcal{S} = \mathcal{F} \setminus \mathcal{C}$ 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 [R87,...]
- How to compute MUSes & MCSes **efficiently** with SAT oracles?

Why it matters?

- Analysis of over-constrained systems
  - Model-based diagnosis [R87,...]
    - 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 $\neg\text{SAT}(\mathcal{M} \setminus \{c\})$ **then**
            $\mathcal{M} \leftarrow \mathcal{M} \setminus \{c\}$ // If $\neg\text{SAT}(\mathcal{M} \setminus \{c\})$, 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 $\neg\text{SAT}(\mathcal{M} \setminus \{c\})$ **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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\text{SAT}(\mathcal{M} \setminus \{c\})$	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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg\text{SAT}(\mathcal{M} \setminus \{c\})$	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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5

Deletion – MUS example

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

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	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)	$(x_5 \vee x_6)$

$\mathcal{M}$	$\mathcal{M} \setminus \{c\}$	$\neg$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
$c_4..c_7$	$c_4 c_6 c_7$	0	Keep c_5
$c_4..c_7$	$c_4 c_5 c_7$	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	$\mathcal{O}(k\ m)$	[PS88,vMW08]
MCS_MUS	$\mathcal{O}(k\ m)$	[BK15]
Deletion-based	$\mathcal{O}(m)$	[CD91,BDTW93]
Linear insertion	$\mathcal{O}(m)$	[MSL'11,BLMS'12]
Dichotomic	$\mathcal{O}(k\ \log(m))$	[HLSB06]
QuickXplain	$\mathcal{O}(k + k\ \log(\frac{m}{k}))$	[J01,J04]
Progression	$\mathcal{O}(k\ \log(1 + \frac{m}{k}))$	[MSJB13,L14]

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

Outline

Minimal Unsatisfiability

Maximum Satisfiability

Additional Exercises

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

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

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

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

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- 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

Recap MaxSAT

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- 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

$x_6 \lor x_2$	$\neg x_6 \lor x_2$	$\neg x_2 \lor x_1$	$\neg x_1$
$\neg x_6 \lor x_8$	$x_6 \lor \neg x_8$	$x_2 \lor x_4$	$\neg x_4 \lor x_5$
$x_7 \lor x_5$	$\neg x_7 \lor x_5$	$\neg x_5 \lor x_3$	$\neg x_3$

- 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

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

- 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

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:

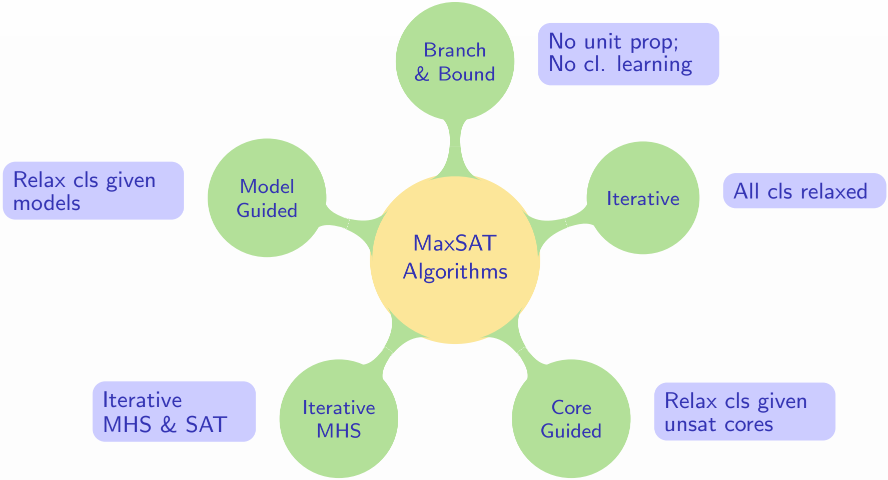
$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - After unit propagation:

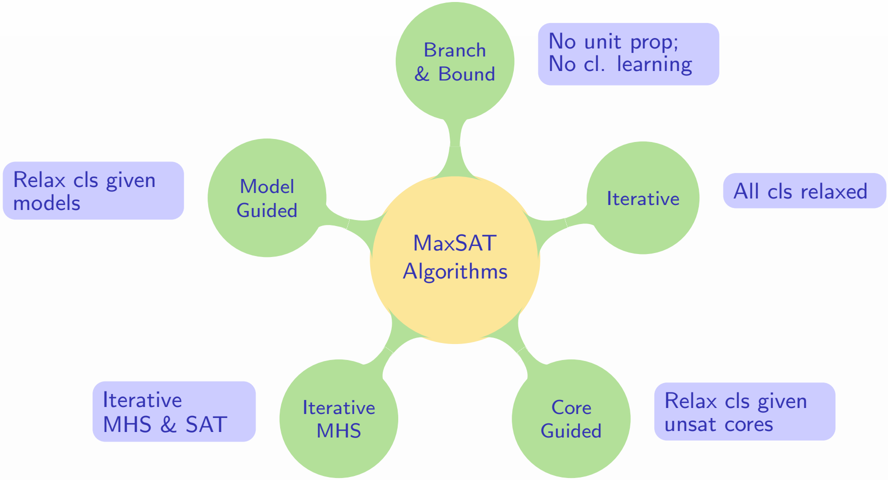
$$\{(x), (\neg x \vee y_1), (\neg x \vee y_2), (\neg y_1 \vee \neg z), (\neg y_2 \vee \neg z), (z)\}$$

  - 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

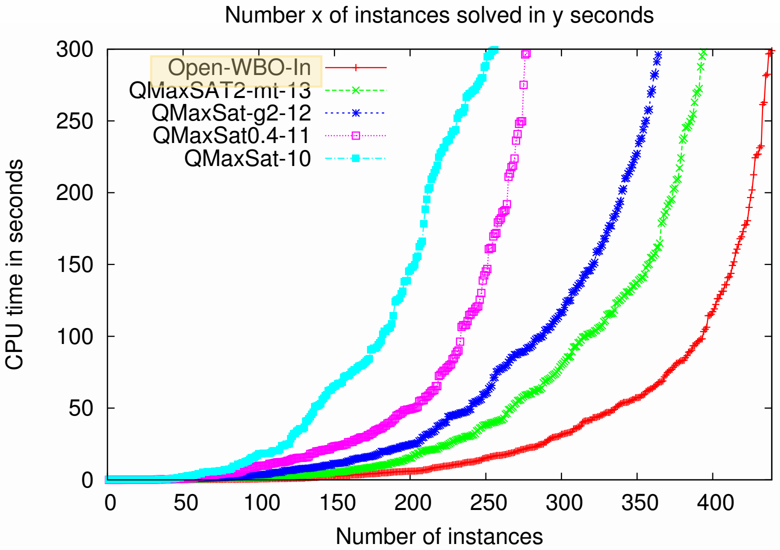


Many MaxSAT approaches



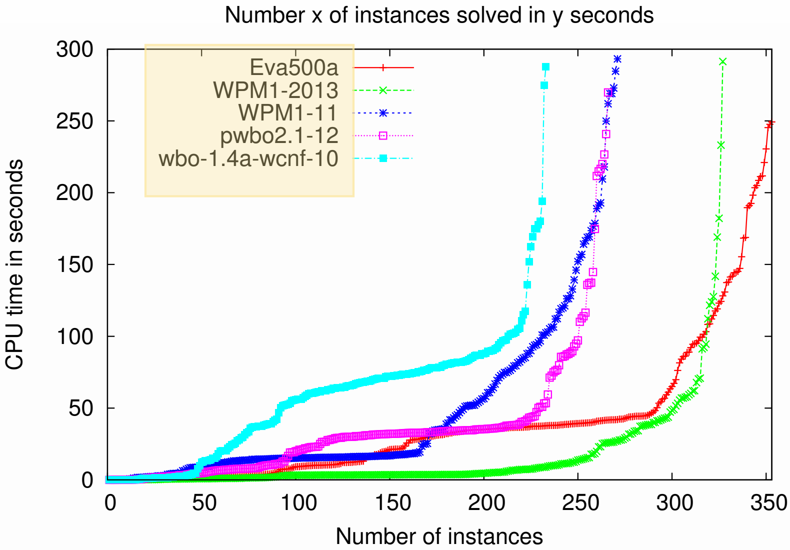
- For practical (**industrial**) instances: **core-guided** approaches are the most effective [MaxSAT14]

Core-guided solver performance – partial



Source: [MaxSAT 2014 organizers]

Core-guided solver performance – weighted partial



Source: [MaxSAT 2014 organizers]

Outline

Minimal Unsatisfiability

Maximum Satisfiability

- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

Additional Exercises

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Example CNF formula

Basic MaxSAT with iterative SAT solving

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

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

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

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

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

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

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

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

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

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

Refine $UB = 3$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

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

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

Basic MaxSAT with iterative SAT solving

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

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

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

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

Refine $UB = 2$

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

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

Formula is UNSAT; terminate

Basic MaxSAT with iterative SAT solving

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

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

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

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

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

Basic MaxSAT with iterative SAT solving

$x_6 \vee x_2 \vee r_1$	$\neg x_6 \vee x_2 \vee r_2$	$\neg x_2 \vee x_1 \vee r_3$	$\neg x_1 \vee r_4$
$\neg x_6 \vee x_8 \vee r_5$	$x_6 \vee \neg x_8 \vee r_6$	$x_2 \vee x_4 \vee r_7$	$\neg x_4 \vee x_5 \vee r_8$
$x_7 \vee x_5 \vee r_9$	$\neg x_7 \vee x_5 \vee r_{10}$	$\neg x_5 \vee x_3 \vee r_{11}$	$\neg x_3 \vee r_{12}$

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

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

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

All (possibly many) soft clauses relaxed

Outline

Minimal Unsatisfiability

Maximum Satisfiability
- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

Additional Exercises

MSU3 core-guided algorithm

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

Example CNF formula

MSU3 core-guided algorithm

$x_6 \vee x_2$	$\neg x_6 \vee x_2$	$\neg x_2 \vee x_1$	$\neg x_1$
$\neg x_6 \vee x_8$	$x_6 \vee \neg x_8$	$x_2 \vee x_4$	$\neg x_4 \vee x_5$
$x_7 \vee x_5$	$\neg x_7 \vee x_5$	$\neg x_5 \vee x_3$	$\neg x_3$

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

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

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

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

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

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

MSU3 core-guided algorithm

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

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

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

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

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

MSU3 core-guided algorithm

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

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

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

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

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

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

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

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

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

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

AtMost$k$/PB constraints used

Some clauses not relaxed

Relaxed soft clauses become **hard**

Outline

Minimal Unsatisfiability

Maximum Satisfiability

- Iterative SAT Solving
- Core-Guided Algorithms
- Minimum Hitting Sets

Additional Exercises

MHS approach for MaxSAT

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

- 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

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

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

MHS approach for MaxSAT

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

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

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

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

- 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

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

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

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

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

- 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

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

- 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

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

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

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

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

- 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

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

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

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

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

- 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

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

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

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

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

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

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

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

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

MHS approach for MaxSAT

$c_1 = x_6 \vee x_2$	$c_2 = \neg x_6 \vee x_2$	$c_3 = \neg x_2 \vee x_1$	$c_4 = \neg x_1$
$c_5 = \neg x_6 \vee x_8$	$c_6 = x_6 \vee \neg x_8$	$c_7 = x_2 \vee x_4$	$c_8 = \neg x_4 \vee x_5$
$c_9 = x_7 \vee x_5$	$c_{10} = \neg x_7 \vee x_5$	$c_{11} = \neg x_5 \vee x_3$	$c_{12} = \neg x_3$

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

- 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***	[e.g. LBP10]
Binary search	Linear*	[e.g. FM06]
FM/WMSU1/WPM1	Exponential**	[FM06,MSM08,MMSP09,ABL09a,ABGL12]
WPM2	Exponential**	[ABL10,ABGL13]
Bin-Core-Dis	Linear	[HMMS11,MHMS12]
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
  - Soft cardinality constraints (OLL)
  - MaxSAT resolution
  - ...

Outline

Minimal Unsatisfiability

Maximum Satisfiability

Additional Exercises

Exercise – How many MCSes & MUSes can there be?

- Give example showing that lower bound on largest number of MCSes is exponential on formula size
  - Hint: Simply suggest formula with exponentially large number of MaxSAT solutions

Exercise – How many MCSes & MUSes can there be?

- Give example showing that lower bound on largest number of MCSes is exponential on formula size
  - Hint: Simply suggest formula with exponentially large number of MaxSAT solutions

- Give example showing that lower bound on largest number of MUSes is exponential on formula size

Solution – number of MCSes

$$\begin{array}{cc} (x_1) & (\neg x_1) \\ (x_2) & (\neg x_2) \\ \dots & \dots \\ (x_n) & (\neg x_n) \end{array}$$

Solution – number of MCSes

$$\begin{array}{cc} (x_1) & (\neg x_1) \\ (x_2) & (\neg x_2) \\ \dots & \dots \\ (x_n) & (\neg x_n) \end{array}$$

- For each $i = 1, \dots, n$ either pick (x_i) or $(\neg x_i)$, i.e. 2 cases

Solution – number of MCSes

$$\begin{array}{cc} (x_1) & (\neg x_1) \\ (x_2) & (\neg x_2) \\ \dots & \dots \\ (x_n) & (\neg x_n) \end{array}$$

- For each $i = 1, \dots, n$ either pick (x_i) or $(\neg x_i)$, i.e. 2 cases
- Thus, 2^n MCSes

Solutions – number of MUSes I

$$(\neg x_1) \wedge (x_1 \vee z_1)$$
$$(\neg y_1) \wedge (y_1 \vee z_1)$$
$$\ldots$$
$$(\neg x_n) \wedge (x_n \vee z_n)$$
$$(\neg y_n) \wedge (y_n \vee z_n)$$

$$(\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n)$$

Solutions – number of MUSes I

$$(\neg x_1) \wedge (x_1 \vee z_1)$$
$$(\neg y_1) \wedge (y_1 \vee z_1)$$
$$\ldots \qquad (\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n)$$
$$(\neg x_n) \wedge (x_n \vee z_n)$$
$$(\neg y_n) \wedge (y_n \vee z_n)$$

- For each $i = 1, \ldots, n$ either resolve away x_i or y_i, i.e. 2 cases

Solutions – number of MUSes I

$$
\begin{array}{ll}
(\neg x_1) \wedge (x_1 \vee z_1) & \\
(\neg y_1) \wedge (y_1 \vee z_1) & \\
\ldots & (\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n) \\
(\neg x_n) \wedge (x_n \vee z_n) & \\
(\neg y_n) \wedge (y_n \vee z_n) &
\end{array}
$$

- For each $i = 1, \ldots, n$ either resolve away x_i or y_i, i.e. 2 cases
- Thus, 2^n MUSes

Solutions – number of MUSes I

$$
\begin{array}{ll}
(\neg x_1) \wedge (x_1 \vee z_1) & \\
(\neg y_1) \wedge (y_1 \vee z_1) & \\
\ldots & (\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n) \\
(\neg x_n) \wedge (x_n \vee z_n) & \\
(\neg y_n) \wedge (y_n \vee z_n) &
\end{array}
$$

- For each $i = 1, \ldots, n$ either resolve away x_i or y_i, i.e. 2 cases
- Thus, 2^n MUSes
- But, there exist formulas with more MUSes. **How?**

Solutions – number of MUSes II

$$(\neg x_1) \wedge (\neg x_2) \wedge \ldots \wedge (\neg x_r)$$
$$(x_1 \vee z_1) \wedge (x_2 \vee z_1) \wedge \ldots \wedge (x_r \vee z_1)$$
$$(x_1 \vee z_2) \wedge (x_2 \vee z_2) \wedge \ldots \wedge (x_r \vee z_2)$$
$$\ldots$$
$$(x_1 \vee z_n) \wedge (x_2 \vee z_n) \wedge \ldots \wedge (x_r \vee z_n)$$
$$(\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n)$$

Solutions – number of MUSes II

$$
\begin{array}{l}
(\neg x_1) \wedge (\neg x_2) \wedge \ldots \wedge (\neg x_r) \\
(x_1 \vee z_1) \wedge (x_2 \vee z_1) \wedge \ldots \wedge (x_r \vee z_1) \\
(x_1 \vee z_2) \wedge (x_2 \vee z_2) \wedge \ldots \wedge (x_r \vee z_2) \\
\ldots \\
(x_1 \vee z_n) \wedge (x_2 \vee z_n) \wedge \ldots \wedge (x_r \vee z_n) \\
(\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n)
\end{array}
$$

- There are r^n MUSes

Solutions – number of MUSes II

$$
\begin{aligned}
&(\neg x_1) \wedge (\neg x_2) \wedge \ldots \wedge (\neg x_r)\\
&(x_1 \vee z_1) \wedge (x_2 \vee z_1) \wedge \ldots \wedge (x_r \vee z_1)\\
&(x_1 \vee z_2) \wedge (x_2 \vee z_2) \wedge \ldots \wedge (x_r \vee z_2)\\
&\ldots\\
&(x_1 \vee z_n) \wedge (x_2 \vee z_n) \wedge \ldots \wedge (x_r \vee z_n)\\
&(\neg z_1 \vee \neg z_2 \vee \ldots \vee \neg z_n)
\end{aligned}
$$

- There are r^n MUSes
- Upper bound by Sperner's theorem: $C(m, \lfloor \frac{m}{2} \rfloor)$

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle
    - How? Simply call a SAT solver

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle
    - How? Simply call a SAT solver
  - Iteratively (and randomly) punch holes in the Sudoku grid
    - If Sudoku puzzle is still valid, i.e. number of solutions is 1, then repeat loop
    - Otherwise, output Sudoku grid with most recently punched hole removed

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle
    - How? Simply call a SAT solver
  - Iteratively (and randomly) punch holes in the Sudoku grid
    - If Sudoku puzzle is still valid, i.e. number of solutions is 1, then repeat loop
    - Otherwise, output Sudoku grid with most recently punched hole removed
  - How many SAT oracles calls?

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle
    - How? Simply call a SAT solver
  - Iteratively (and randomly) punch holes in the Sudoku grid
    - If Sudoku puzzle is still valid, i.e. number of solutions is 1, then repeat loop
    - Otherwise, output Sudoku grid with most recently punched hole removed
  - How many SAT oracles calls?
    - Linear number of calls on number of cells in Sudoku puzzle

Exercise – Sudoku puzzle generator

- Randomly generate valid Sudoku puzzles starting from a filled grid
- How?
  - Pick a known valid Sudoku puzzle
  - Complete the Sudoku puzzle
    - How? Simply call a SAT solver
  - Iteratively (and randomly) punch holes in the Sudoku grid
    - If Sudoku puzzle is still valid, i.e. number of solutions is 1, then repeat loop
    - Otherwise, output Sudoku grid with most recently punched hole removed
  - How many SAT oracles calls?
    - Linear number of calls on number of cells in Sudoku puzzle
  - Can we do better?

Some final notes

- SAT is a low-level, but very powerful problem solving paradigm
- There is an ongoing revolution on problem solving with SAT oracles
- The use of SAT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.

Some final notes

- SAT is a low-level, but very powerful problem solving paradigm
- There is an ongoing revolution on problem solving with SAT oracles
- The use of SAT oracles is impacting problem solving for many different complexity classes
  - With well-known representative problems, e.g. QBF, #SAT, etc.
- Many fascinating research topics out there !

Links for tools

- SAT solvers:
  - minisat: https://github.com/niklasso/minisat
  - glucose: http://www.labri.fr/perso/lsimon/glucose/
- MaxSAT solvers:
  - MSCG: http://logos.ucd.ie/web/doku.php?id=mscg
  - OpenWBO: http://sat.inesc-id.pt/open-wbo/
  - MaxHS: http://www.maxhs.org
- MCS extractors:
  - mcsXL: http://logos.ucd.ie/wiki/doku.php?id=mcsxl
  - LBX: http://logos.ucd.ie/wiki/doku.php?id=lbx
  - MCSls: http://logos.ucd.ie/wiki/doku.php?id=mcsls
- MUS extractors:
  - MUSer: http://logos.ucd.ie/wiki/doku.php?id=muser
- Many other tools available from:
  http://logos.ucd.ie/wiki/doku.php?id=soft

Thank You