Introduction to Satisfiability

Jinbo Huang

NICTA and Australian National University

Overview

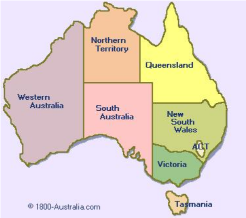
- Satisfiability
- Algorithms
- Applications
- Extensions

Satisfiability

Broadly, can set of constraints be satisfied?

Example: neighbors have different colors on map

$WA \neq NT, WA \neq SA, \ldots$



Boolean Satisfiability (SAT)

All variables are Boolean (true/false)

Constraints are *clauses*

- $A \vee B, \overline{B} \vee C \vee D, \ldots$
- disjunction (logical OR) of *literals*
- literal: variable or its negation
- set of clauses: conjunctive normal form (CNF)

Clause satisfied when ≥ 1 literal is true

Why CNF?

Any Boolean formula can be put in CNF

With auxilary variables, polynomial-size CNF preserves satisfiability

Modeling is relatively natural

- problem broken into parts/steps
- model each part/step as (small) set of clauses

Why Boolean: Versatility

Can model many types of problems

- planning/scheduling, spatial/temporal reasoning, hardware/software verification, test generation, diagnosis, bioinformatics, ...

$WA \neq NT$ translates into
$WA_R \rightarrow \overline{NT_R}, WA_B \rightarrow \overline{NT_B}, WA_G \rightarrow \overline{NT_G}, \ldots$

($X \rightarrow Y$ is short for $\overline{X} \vee Y$)

Encoding can be larger, but reasoning not necessarily slower

Why Boolean: NP-Completeness

All of NP translate to SAT in polynomial time

- $p \in$ NP $\Rightarrow p$ is solved by an algorithm
- algo. works by moving from one state to next
  - content of memory (computer) or tape (Turing machine)
  - encode state with set of Boolean variables
  - size of state: polynomial
  - # of states needed: polynomial
- moves determined by instructions in algo.
  - encode instructions with Boolean formulas
  - # of sets needed: polynomial
- write formula that is satisfiable iff algorithm says YES

Why Boolean: Practical Solvers

Often efficient & scalable on real-world problems

- often handle millions of clauses

Some techniques not readily applicable to constraint solvers

- fast propagation (literal watching)
- learning & restarts (exponential boost)
- decision heuristic (driven by learning)

Algorithms for SAT

Clause learning

- top 3 in Application, SAT Competition 2009: PRECOSAT, GLUCOSE, LySAT
- 2 of top 3 in Crafted: CLASP, MINISAT

Local search

- top 3 in Random Satisfiable: TNM, GNOVELTY+2, HYBRIDGM3

Other algorithms

- lookahead, solver portfolios

Algorithm: Enumeration

$\{\overline{A} \vee B, \overline{B} \vee C\}$

Is formula satisfiable?

Algorithm: Enumeration

$\{\overline{A} \vee B, \overline{B} \vee C\}$

Is formula satisfiable?

8 assignments to ABC: enumerate & check, until *model* found, e.g., $A = B = C = t$

Algorithm: Enumeration

$\{\overline{A} \vee B, \overline{B} \vee C\}$

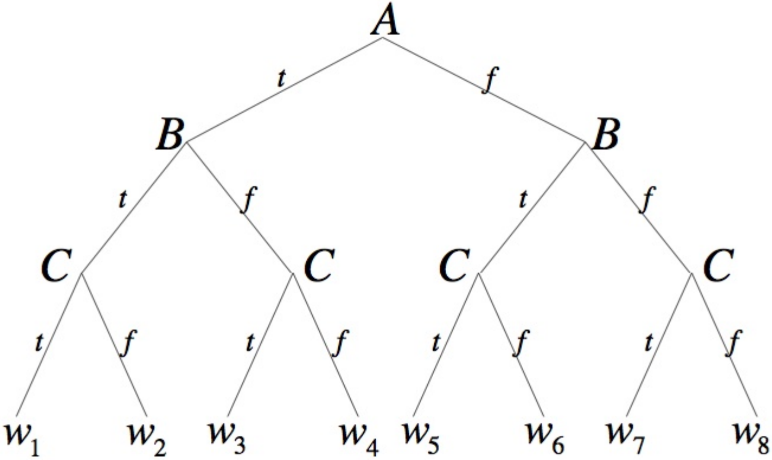
Is formula satisfiable?

8 assignments to ABC: enumerate & check, until *model* found, e.g., $A = B = C = t$

Exponential in # of variables in worst case (fine, but can try to do better in average case)

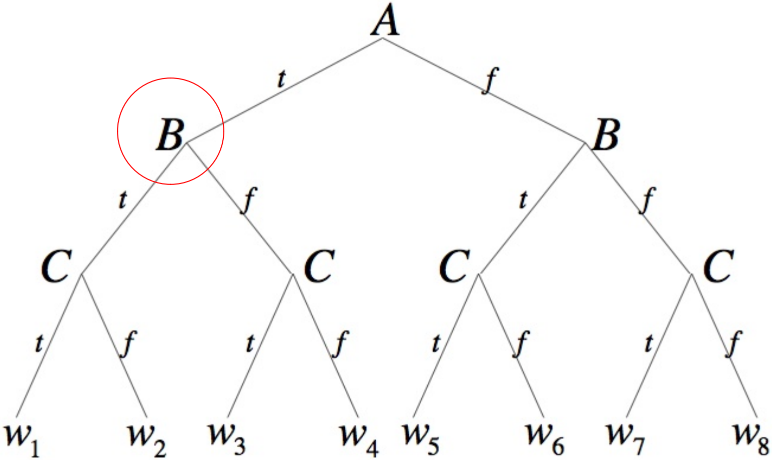
Algorithm: Enumeration by Search

$\{\overline{A} \vee B, \overline{B} \vee C\}$



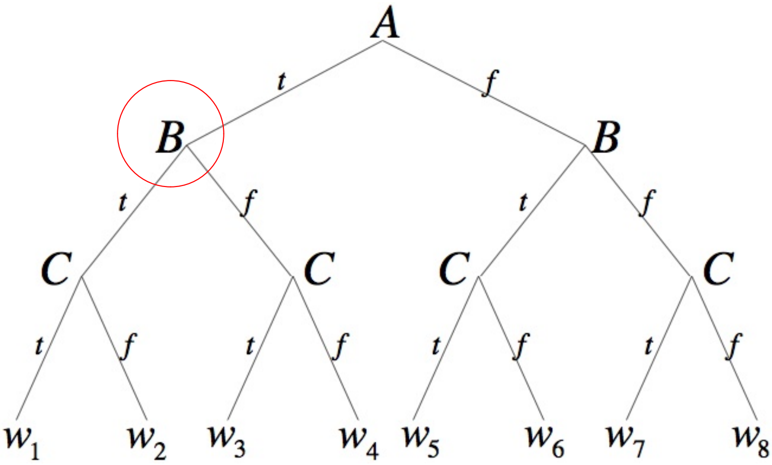
Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A$



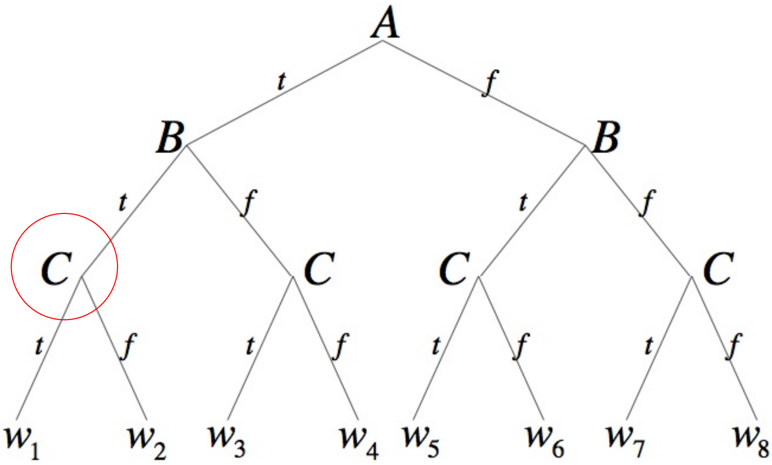
Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}$



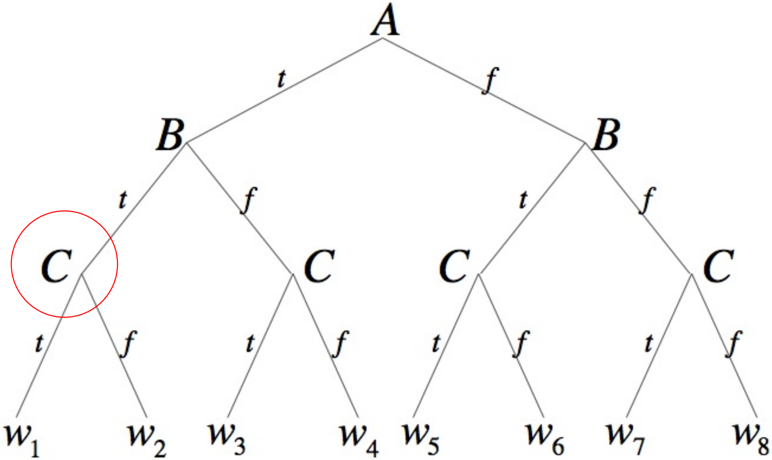
Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}|_B$



Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}|_B \quad \{C\}$



Conditioning

$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \quad \{B, \overline{B} \vee C\}|_B \quad \{C\}$

Simplifies formula

- false literal disappears from clause
- true literal makes clause disappear

Leaves of search tree

- empty clause generated: formula falsified
- all clauses gone: formula satisfied

Not necessarily 2^n leaves, even in case of UNSAT

Early Backtracking

Backtrack as soon as empty clause generated

Early Backtracking

Backtrack as soon as empty clause generated

Can we do even better?

Early Backtracking

Backtrack as soon as empty clause generated

Can we do even better?

What about unit clauses?
$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \;\; \{B, \overline{B} \vee C\}$

Early Backtracking

Backtrack as soon as empty clause generated

Can we do even better?

What about unit clauses?
$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \ \{B, \overline{B} \vee C\}$

B must be true, no need for two branches

Early Backtracking

Backtrack as soon as empty clause generated

Can we do even better?

What about unit clauses?
$\{\overline{A} \vee B, \overline{B} \vee C\}|_A \ \{B, \overline{B} \vee C\}$

B must be true, no need for two branches

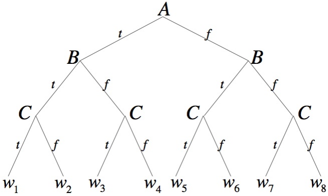
Setting $B = t$ may lead to more unit clauses, repeat till no more (or till empty clause)

Known as unit propagation

Algorithm: DPLL

Same kind of search tree, each node augmented with unit propagation

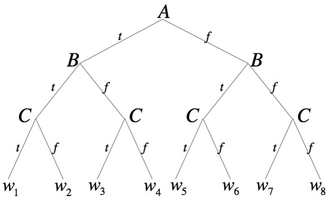
- multiple assignments in one level
  - decision & implications
- may not need n levels to reach leaf



Algorithm: DPLL

Same kind of search tree, each node augmented with unit propagation

- multiple assignments in one level
  - decision & implications
- may not need n levels to reach leaf



What (completely) determines search tree?

DPLL: Variable Ordering

Can have huge impact on efficiency

Example: unit propagation lookahead, as in SATZ

- short clauses are good, more likely to result in unit propagation
- tentatively try each variable, count new binary clauses generated
- select variable with highest score:
  $w(X) \cdot w(\overline{X}) \cdot 1024 + w(X) + w(\overline{X})$

Generally different orders down different branches: dynamic ordering

Enhancement

Given variable ordering, search tree is fixed

How can we possibly reduce search tree further?

Enhancement

Given variable ordering, search tree is fixed

How can we possibly reduce search tree further?

Backtrack earlier

Enhancement

Given variable ordering, search tree is fixed

How can we possibly reduce search tree further?

Backtrack earlier

Backtracking occurs (only) when empty clause generated

Empty clause generated (only) by unit propagation

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

What clauses to add?

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

What clauses to add?

- not already in CNF

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

What clauses to add?

- not already in CNF
- logically implied by CNF (or correctness lost)

Empowering Unit Propagation

Unit propagation determined by set of clauses

More clauses $\Rightarrow$ (potentially) more propagation, earlier empty clause (backtrack), smaller search tree

What clauses to add?

- not already in CNF
- logically implied by CNF (or correctness lost)
- empower UP

Clause Learning

A, B
B, C
$\overline{A}, \overline{X}, Y$
$\overline{A}, X, Z$
$\overline{A}, \overline{Y}, Z$
$\overline{A}, X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$

Clause Learning

$\Delta|_A$

A, B
B, C
$\overline{A}, \overline{X}, Y$
$\overline{A}, X, Z$
$\overline{A}, \overline{Y}, Z$
$\overline{A}, X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$

Clause Learning

	$\Delta\|_A$
A, B	
B, C	B, C
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

Clause Learning

	$\Delta\vert_A$	$\Delta\vert_{A,B}$
A, B		
B, C	B, C	
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	
$\overline{A}, X, Z$	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	

Clause Learning

	$\Delta\vert_A$	$\Delta\vert_{A,B}$
A, B		
B, C	B, C	
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

Clause Learning

	$\Delta\vert_A$	$\Delta\vert_{A,B}$	$\Delta\vert_{A,B,C}$
A, B			
B, C	B, C		
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	
$\overline{A}, X, Z$	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	

Clause Learning

	$\Delta\|_A$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$
A, B			
B, C	B, C		
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z	X, Z	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

Clause Learning

	$\Delta\|_{A}$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	

Clause Learning

	$\Delta\|_A$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

Clause Learning

	$\Delta\|_A$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$

Clause Learning

	$\Delta\vert_{A}$	$\Delta\vert_{A,B}$	$\Delta\vert_{A,B,C}$	$\Delta\vert_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$
- B, C irrelevant

Clause Learning

	$\Delta\vert_A$	$\Delta\vert_{A,B}$	$\Delta\vert_{A,B,C}$	$\Delta\vert_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$
- B, C irrelevant: $\Delta|_A \Rightarrow \overline{X}$

Clause Learning

	$\Delta\|_A$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$
- B, C irrelevant: $\Delta|_A \Rightarrow \overline{X}$
- What clause would have allowed UP to derive $\overline{X}$ in level 0?

Clause Learning

	$\Delta\|_A$	$\Delta\|_{A,B}$	$\Delta\|_{A,B,C}$	$\Delta\|_{A,B,C,X}$
A, B				
B, C	B, C			
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	$\overline{X}, Y$	Y
$\overline{A}, X, Z$	X, Z	X, Z	X, Z	
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	$X, \overline{Z}$	
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$

- Conflict in level 3: $\Delta|_{A,B,C} \Rightarrow \overline{X}$
- B, C irrelevant: $\Delta|_A \Rightarrow \overline{X}$
- What clause would have allowed UP to derive $\overline{X}$ in level 0? $\overline{A} \vee \overline{X}$ $(A \rightarrow \overline{X})$

Clause Learning

A, B
B, C
$\overline{A}, \overline{X}, Y$
$\overline{A}, X, Z$
$\overline{A}, \overline{Y}, Z$
$\overline{A}, X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$
$\color{red}{\overline{A}, \overline{X}}$

Clause Learning

	$\Delta\|_A$
A, B	
B, C	B, C
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$
$\overline{A}, \overline{X}$	$\overline{X}$

Clause Learning

$$
\begin{array}{ll}
 & \Delta|_A \\
A, B & \\
B, C & B, C \\
\overline{A}, \overline{X}, Y & \overline{X}, Y \\
\overline{A}, X, Z & X, Z \\
\overline{A}, \overline{Y}, Z & \overline{Y}, Z \\
\overline{A}, X, \overline{Z} & X, \overline{Z} \\
\overline{A}, \overline{Y}, \overline{Z} & \overline{Y}, \overline{Z} \\
\overline{A}, \overline{X} & \overline{X}
\end{array}
$$

$\overline{A} \vee \overline{X}$ satisfies criteria

- not in CNF
- implied by CNF
- empowers UP

Learn clause in level 3

Backtrack to level 0, start over

Clause Learning

	$\Delta\vert_A$
A, B	
B, C	B, C
$\overline{A}, \overline{X}, Y$	$\overline{X}, Y$
$\overline{A}, X, Z$	X, Z
$\overline{A}, \overline{Y}, Z$	$\overline{Y}, Z$
$\overline{A}, X, \overline{Z}$	$X, \overline{Z}$
$\overline{A}, \overline{Y}, \overline{Z}$	$\overline{Y}, \overline{Z}$
$\overline{A}, \overline{X}$	$\overline{X}$

$\overline{A} \vee \overline{X}$ satisfies criteria

- not in CNF
- implied by CNF
- empowers UP

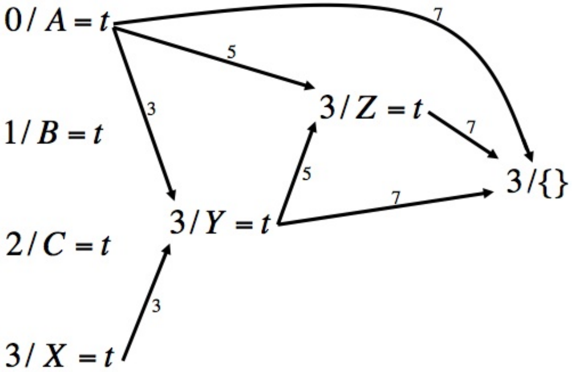
Learn clause in level 3

Backtrack to level 0, start over

How to learn? How far to backtrack?

Implication Graph

$1 : A, B$
$2 : B, C$
$3 : \overline{A}, \overline{X}, Y$
$4 : \overline{A}, X, Z$
$5 : \overline{A}, \overline{Y}, Z$
$6 : \overline{A}, X, \overline{Z}$
$7 : \overline{A}, \overline{Y}, \overline{Z}$

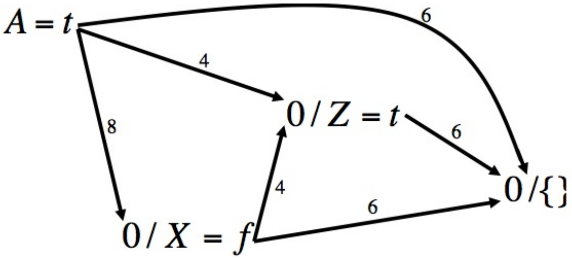


Implication Graph

$1 : A, B$
$2 : B, C$
$3 : \overline{A}, \overline{X}, Y$
$4 : \overline{A}, X, Z$
$5 : \overline{A}, \overline{Y}, Z$
$6 : \overline{A}, X, \overline{Z}$
$7 : \overline{A}, \overline{Y}, \overline{Z}$
$8 : \overline{A}, \overline{X}$

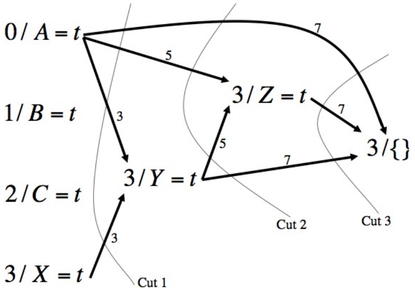
Implication Graph

$1: A, B$
$2: B, C$
$3: \overline{A}, \overline{X}, Y$
$4: \overline{A}, X, Z$
$5: \overline{A}, \overline{Y}, Z$
$6: \overline{A}, X, \overline{Z}$
$7: \overline{A}, \overline{Y}, \overline{Z}$
$8: \overline{A}, \overline{X}$



Conflict Analysis

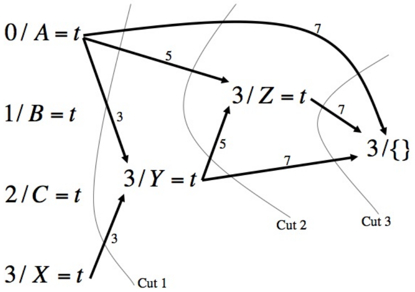
$1 : A, B$
$2 : B, C$
$3 : \overline{A}, \overline{X}, Y$
$4 : \overline{A}, X, Z$
$5 : \overline{A}, \overline{Y}, Z$
$6 : \overline{A}, X, \overline{Z}$
$7 : \overline{A}, \overline{Y}, \overline{Z}$



- Cut: roots (decisions) on one side, sink (contradiction) on other
- Arrows across cut together responsible for contradiction
- Conflict set: tail points of arrows

Conflict Set

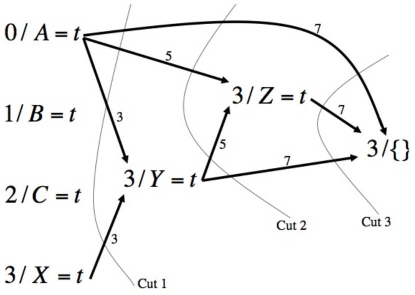
1 : A, B
2 : B, C
3 : $\overline{A}, \overline{X}, Y$
4 : $\overline{A}, X, Z$
5 : $\overline{A}, \overline{Y}, Z$
6 : $\overline{A}, X, \overline{Z}$
7 : $\overline{A}, \overline{Y}, \overline{Z}$



- Cut 1: $\{A, X\}$
- Cut 2: $\{A, Y\}$
- Cut 3: $\{A, Y, Z\}$

Conflict Clause

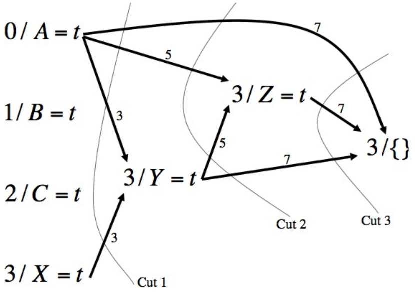
$1: A, B$
$2: B, C$
$3: \overline{A}, \overline{X}, Y$
$4: \overline{A}, X, Z$
$5: \overline{A}, \overline{Y}, Z$
$6: \overline{A}, X, \overline{Z}$
$7: \overline{A}, \overline{Y}, \overline{Z}$



- Cut 1: $\{A, X\} \Rightarrow \overline{A} \vee \overline{X}$
- Cut 2: $\{A, Y\} \Rightarrow \overline{A} \vee \overline{Y}$
- Cut 3: $\{A, Y, Z\} \Rightarrow \overline{A} \vee \overline{Y} \vee \overline{Z}$ (existing)

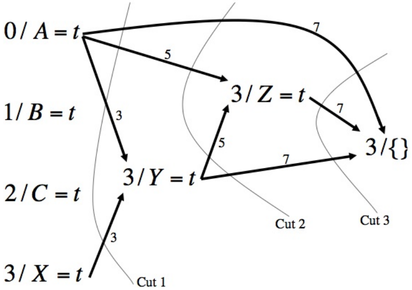
Conflict Clause

$1 : A, B$
$2 : B, C$
$3 : \overline{A}, \overline{X}, Y$
$4 : \overline{A}, X, Z$
$5 : \overline{A}, \overline{Y}, Z$
$6 : \overline{A}, X, \overline{Z}$
$7 : \overline{A}, \overline{Y}, \overline{Z}$



- Cut 1: $\{A, X\} \quad \Rightarrow \overline{A} \vee \overline{X}$
- Cut 2: $\{A, Y\} \quad \Rightarrow \overline{A} \vee \overline{Y}$
- Which clause to learn?

Unique Implication Point (UIP)



Prefer shorter explanation

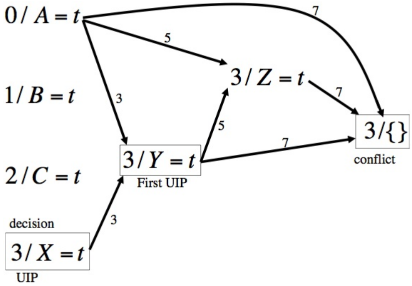
- shorter clause closer to unit, more empowering

Never need >1 node from latest level

- latest decision $+$ history always suffices

UIP: lies on all paths from decision to contradiction

Unique Implication Point (UIP)



Prefer shorter explanation

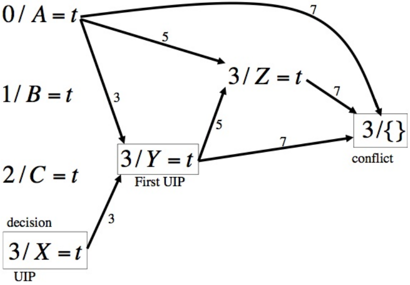
- shorter clause closer to unit, more empowering

Never need > 1 node from latest level

- latest decision $+$ history always suffices

UIP: lies on all paths from decision to contradiction

1-UIP Learning



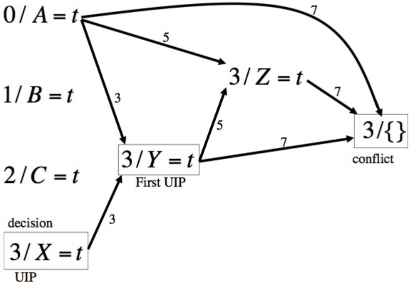
Work from sink backwards

Stop when conflict set includes a UIP, and no other nodes, of latest level: 1-UIP clause $(\overline{A} \vee \overline{Y})$

- 2-UIP, 3-UIP, ..., All-UIP

Empirically shown effective, most common choice

Backtracking to Assertion Level



Learned clause: $\overline{A} \vee \overline{Y}$

- becomes unit ($\overline{Y}$) when erasing current level
- asserting clause: UP will assert $\overline{Y}$ (empowerment)

Backtracking to Assertion Level

Backtrack as far as possible, as long as UP remains empowered

Assertion level: 2nd highest level in learned clause, or -1 if learned clause is unit

- $A_0 \vee \overline{B}_1 \vee C_1 \vee X_4$: $aLevel = 1$
- X_4: $aLevel = -1$
- learned unit clause asserted before any decision

Empirically shown effective, most common choice

Clause Learning: Putting It Together

REPEAT

  IF no free variable

    RETURN SAT

  pick free variable X and set either X or $\overline{X}$

  IF contradiction

    IF level < 0

      RETURN UNSAT

    learn clause

    backtrack anywhere learned clause $\neq \emptyset$

Clause Learning: Putting It Together

REPEAT
  IF no free variable
    RETURN SAT
  pick free variable X and set either X or $\overline{X}$
  IF contradiction
    IF level < 0
      RETURN UNSAT
    learn clause
    backtrack anywhere learned clause $\neq \emptyset$

- No more branching, unlike DPLL
- Conflict-driven, repeated probing

Clause Learning: Putting It Together

REPEAT
  IF no free variable
    RETURN SAT
  pick free variable X and set either X or $\overline{X}$
  IF contradiction
    IF level < 0
      RETURN UNSAT
    learn clause
    backtrack anywhere learned clause $\neq \emptyset$

- Completeness?

Clause Learning: Putting It Together

REPEAT
    IF no free variable
        RETURN SAT
    pick free variable X and set either X or $\overline{X}$
    IF contradiction
        IF level < 0
            RETURN UNSAT
        learn clause
        backtrack anywhere learned clause $\neq \emptyset$

- Will terminate because learned clause must be new, $|$clauses$|$ finite

Clause Learning: Putting It Together

REPEAT
  IF no free variable
    RETURN SAT
  pick free variable X and set either X or $\overline{X}$
  IF contradiction
    IF level < 0
      RETURN UNSAT
    learn clause
    backtrack anywhere learned clause $\neq \emptyset$

- Components: decision heuristic, learning method, backtracking method

Decision Heuristic: Popular Ideas

Learned conflict clause summarizes cause of failure

Try to satisfy conflict clauses

- helps eventually satisfy whole CNF if SAT
- helps terminate early if UNSAT

Maintain occurrence count for each literal

- increment on learning new clause
- periodically shrink all counts: recent activity more relevant

Pick variable with highest count ($+\&-$ combined)

- set to same value it had last: progress saving

Progress Saving

Backtracking erases multiple levels of assignments

Some of those may have satisfied parts of CNF

Reusing assignments helps avoid having to rediscover those partial solutions

Restarts: Special Case of Backtracking

Re-make decisions in light of new clauses

Restarts: Special Case of Backtracking

Re-make decisions in light of new clauses

- restart at predetermined intervals
- restart based on current search activity

Restarts: Special Case of Backtracking

Re-make decisions in light of new clauses

- restart at predetermined intervals
- restart based on current search activity

Important empirically

- solvers with no restarts uncompetitive
- performance sensitive to restart policy

Important theoretically

- clause learning more powerful than DPLL, proof relies on restarts

Efficient Unit Propagation

Need to detect unit clauses

Naively, keep track of clause lengths: when setting X, decrement lengths of clauses that contain $\overline{X}$

- inefficient when CNF is large

Pick 2 literals to watch in each clause

- watch A, B in $A \vee B \vee \overline{C} \vee D$
- clause cannot be unit unless $\overline{A}$ or $\overline{B}$ is set
- do nothing when C or D is assigned

Scales to millions of clauses in practice

Clause Learning: Summary

Fundamentally different search scheme from DPLL

- no branching
- sequence of decisions, learn, backtrack, repeat
- theoretically more powerful than DPLL

What determines search behavior

- methods for decision, learning, backtracking (including restarts)
- popular choices: literal activity + progress saving, 1-UIP learning, backtracking to assertion level (various restart policies being explored)

Clause Learning and Resolution

Resolution p-simulates clause learning

- Each learned clause obtained by resolution
- At termination, resolution proof can be extracted (in polytime)

Clause Learning and Resolution

Clause learning p-simulates resolution

- Have clause learning absorb interesting clauses of resolution proof (in polytime)
  - interesting: 1-empowering, 1-provable
  - absorb: render it useless (not 1-empowering)
- Interesting clause always exists unless Δ 1-inconsistent
- Hence clause learning will terminate after absorbing all interesting clauses

Hard Problems for Resolution: Pigeonhole

P_{ij}: pigeon i in hole j

$P_{11} \vee P_{12}$, $P_{21} \vee P_{22}$, $P_{31} \vee P_{32}$
$\neg P_{11} \vee \neg P_{21}$, $\neg P_{21} \vee \neg P_{31}$, $\neg P_{11} \vee \neg P_{31}$
$\neg P_{12} \vee \neg P_{22}$, $\neg P_{22} \vee \neg P_{32}$, $\neg P_{12} \vee \neg P_{32}$

No polynomial resolution proof for PH_n

Extended Resolution

Introduce *new variables* into proof

Extension: $x \leftrightarrow \phi$

- ϕ: formula over existing variables
- Suffices to restrict ϕ to $l_1 \vee l_2$

Otherwise same as resolution

Extended Resolution

Can simulate (compact) proof by induction

Pigeonhole: No 1-to-1 map from $\{1, \ldots, n\}$ to $\{1, \ldots, n-1\}$

- Base case ($n = 2$): easy
- If $f(i)$ maps $\{1, \ldots, n\}$ to $\{1, \ldots, n-1\}$
- Define $f'(i)$ from $\{1, \ldots, n-1\}$ to $\{1, \ldots, n-2\}$
  - $f'(i) = f(i)$ if $f(i) \neq n-1$
  - $f'(i) = f(n)$ otherwise

Extended Resolution

Induction proof

- If $f(i)$ maps $\{1, \ldots, n\}$ to $\{1, \ldots, n-1\}$
- Define $f'(i)$ from $\{1, \ldots, n-1\}$ to $\{1, \ldots, n-2\}$
  - $f'(i) = f(i)$ if $f(i) \neq n-1$
  - $f'(i) = f(n)$ otherwise

ER proof simulating above

- $\{P_{ij}\}$ describes f (PH_n), introduce $\{Q_{ij}\}$ to describe f' (PH_{n-1})
  - $Q_{ij} \leftrightarrow (P_{ij} \vee (P_{i,n-1} \wedge P_{nj}))$
- $O(n^3)$ resolutions to derive PH_{n-1}
- Repeat until base case PH_2: $R_{11}, R_{21}, \neg R_{11} \vee \neg R_{21}$

Extended Resolution

Strictly more powerful than resolution

Adding Extensions to Clause Learning

How does solver decide?

Adding Extensions to Clause Learning

How does solver decide?

Compare simulation of resolution by solver

- Resolution itself provides no guidance on what clauses to resolve
- Solver uses *probings* as guide
- Reduces search space, retains power

Adding Extensions to Clause Learning

How does solver decide?

Compare simulation of resolution by solver

- Resolution itself provides no guidance on what clauses to resolve
- Solver uses *probings* as guide
- Reduces search space, retains power

Prune space of extensions

Useless Extensions

$x \leftrightarrow l_1 \vee l_2$ useless if $\Delta \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathit{false}$

Useless Extensions

$x \leftrightarrow l_1 \vee l_2$ useless if $\Delta \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathit{false}$

Lemma: Banning them does not affect power of ER

Useless Extensions

$x \leftrightarrow l_1 \vee l_2$ useless if $\Delta \cup \{\overline{l_1}, \overline{l_2}\} \vdash \mathit{false}$

Lemma: Banning them does not affect power of ER

Efficient filtering of useless extensions?

Useless Extensions

Theorem: Solver need only pick $l_1 \vee l_2$ from assignment stack (with negation)

- Due to forced assignments, not all combinations of $l_1 \vee l_2$ possible
- But those would be useless anyway
- Full power of ER retained
- Decision heuristic doubles as guide for extensions

A More Concrete Heuristic

Pick $l_1 \vee l_2$ from learned clause, if length $\geq k$

- Literals in learned clause must come from assignment stack (with negation)

Open question: Does this restrict power of ER

Experiments: Application Benchmarks

Results mixed

However, where it worked, improvement very substantial

- From unsolved to solved (in 5–30 minutes)
- Search tree size reduced by factor of 5–42

Experiments: Crafted Benchmarks

gt-ordering: any partial order on $\{1, \ldots, n\}$ must have maximal element

7 instances, none could be solved by baseline solver

All solved, in 39 seconds

Extended Clause Learning: Summary

Extensions can lead to substantial practical gains

Extension heuristics a promising research direction

Catch: More powerful proof system, harder to find short proof

Local Search

Clause learning poor on random problems

- good at exploiting structure
- little/no structure in random problems

Local Search

Clause learning poor on random problems

- good at exploiting structure
- little/no structure in random problems

Local search

- start with complete assignment
- if CNF satisfied, done; else flip a var, repeat
- incomplete: may not find model, cannot prove unsatisfiability

GSAT

REPEAT MAX-TRIES times
  randomly generate assignment α
  REPEAT MAX-FLIPS times
    IF α satisfies CNF THEN RETURN SAT
    flip variable in α for least falsified clauses
RETURN FAIL

- quickly descends toward better assignment
- spends much time moving "sideways" on a plateau, before exiting into better plateau
- may get stuck in local minimum

Walksat

- flip variable in falsified clause (more focus)
- introduce noise to escape from plateaus

REPEAT MAX-TRIES times
    randomly generate assignment α
    REPEAT MAX-FLIPS times
        IF α satisfies CNF THEN RETURN SAT
        randomly pick falsified clause
        IF $\exists$ "freebie move" THEN do it
        ELSE
            with probability p, flip random var in clause
            else flip var in clause for least "break count"
RETURN FAIL

Phase Transition in Random Problems

k-SAT: k literals per clause

Vary # of (random) clauses for given # of variables

- low ratio: nearly all SAT, easy
- high ratio: nearly all UNSAT, easy
- phase transition ($\approx$ 4.2 for 3-SAT): about half SAT, half UNSAT, hardest

Solver Portfolio

Keep collection of solvers

Train solver selector on large set of instances

Use it to select solver for given instance

1 of top 3 in Crafted, SAT Competition 2009: SATzilla

Applications of SAT

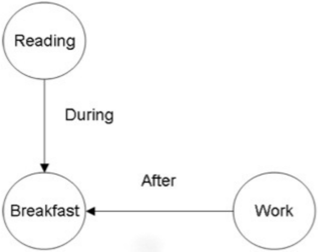
- Qualitative temporal reasoning
- Constraint solving

Qualitative Temporal Reasoning

Reasoning about time intervals (events)

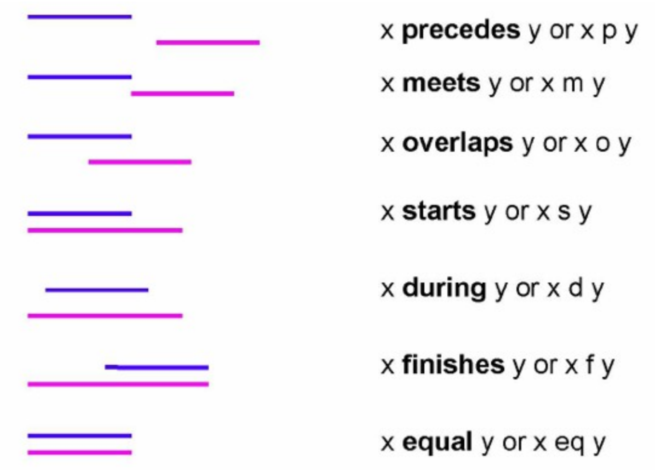
Qualitative: relations between intervals

- not concerned with exact time points



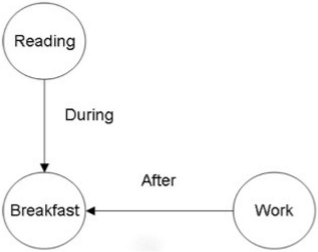
"Peter reads newspaper during breakfast, goes to work after breakfast"

Interval Algebra (IA)



- All have inverse, total of 13 atomic relations

The Reasoning Task



Is given temporal (IA) network satisfiable?

- nodes: variables
- edges: constraints

- infinite domain (all possible intervals on a line)
  - traditional search doesn't work
- $2^{13} = 8192$ possible relations
  - "Peter reads private email before or after work"

Transforming Search Space

Don't search for instantiation of nodes (intervals)

Search for instantiation of edges

- edge: set of atomic relations
- any consistent instantiation of nodes satisfies exactly 1 per edge
- need only search for satisfiable atomic refinement of network

Transforming Search Space

Don't search for instantiation of nodes (intervals)

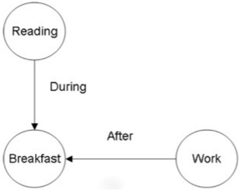
Search for instantiation of edges

- edge: set of atomic relations
- any consistent instantiation of nodes satisfies exactly 1 per edge
- need only search for satisfiable atomic refinement of network

Theorem

Atomic IA network is satisfiable iff path-consistent

Path Consistency

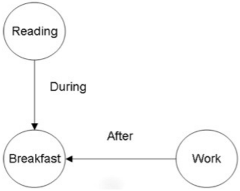


Any consistent assignment for 2 nodes can be extended to consistent assignment for 3rd

$\forall ABC, a, b$, if $A_a \sim B_b$ then $\exists c$ $C_c \sim A_a$ and $C_c \sim B_b$

- reading 7:10–7:20, breakfast 7:00–7:30
- can assign work 8:00–12:00
- missing edge: universal relation

Path Consistency



Any consistent assignment for 2 nodes can be extended to consistent assignment for 3rd

$\forall ABC, a, b$, if $A_a \sim B_b$ then $\exists c$ $C_c \sim A_a$ and $C_c \sim B_b$

- reading 7:10–7:20, work 7:15–12:00
- no way to assign breakfast
- refine (invisible) edge W-R: universal $\longrightarrow$ **after**

Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

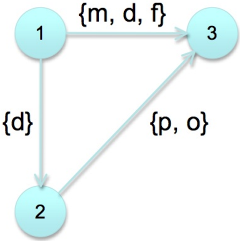
- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$

Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$

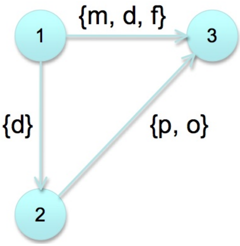


Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$



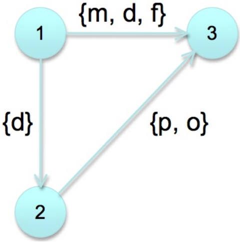
$R_{12d}, R_{23p} \vee R_{23o}, R_{13m} \vee R_{13d} \vee R_{13f}$

Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$



$R_{12d}, R_{23p} \vee R_{23o}, R_{13m} \vee R_{13d} \vee R_{13f}$

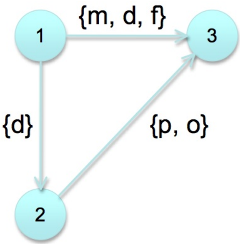
$R_{12d} \wedge R_{23p} \longrightarrow \mathit{false}$

Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$



$R_{12d}, R_{23p} \vee R_{23o}, R_{13m} \vee R_{13d} \vee R_{13f}$

$R_{12d} \wedge R_{23p} \rightarrow \mathit{false}$

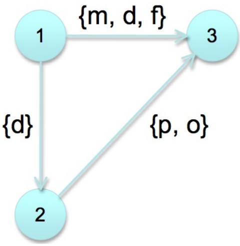
$R_{12d} \wedge R_{23o} \rightarrow (R_{13m} \vee R_{13d})$

Ensuring Path Consistency

Atomic network: $\forall ABC : R_{AC} \in R_{AB} \circ R_{BC}$

Composition

- $\{d\} \circ \{p\} = \{p\}$
- $\{d\} \circ \{o\} = \{p, m, o, s, d\}$



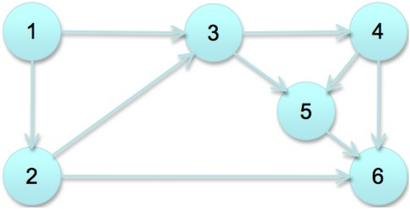
$R_{12d}, R_{23p} \vee R_{23o}, R_{13m} \vee R_{13d} \vee R_{13f}$

$R_{12d} \wedge R_{23p} \rightarrow \mathit{false}$

$R_{12d} \wedge R_{23o} \rightarrow (R_{13m} \vee R_{13d})$

Alternatively, use inequalities between points, better in practice

Solving IA Networks by SAT

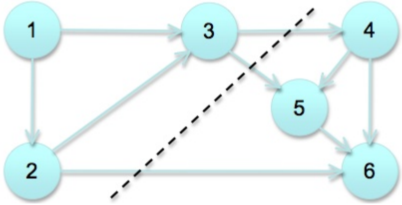


Encode each $\triangle$ in complete graph

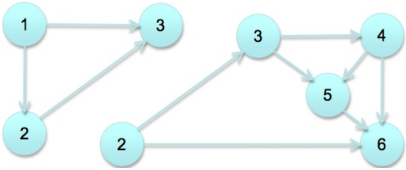
- invisible edges are edges with universal relation
- $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$ triangles

CNF satisfiable iff IA network satisfiable

More Compact Encoding



More Compact Encoding



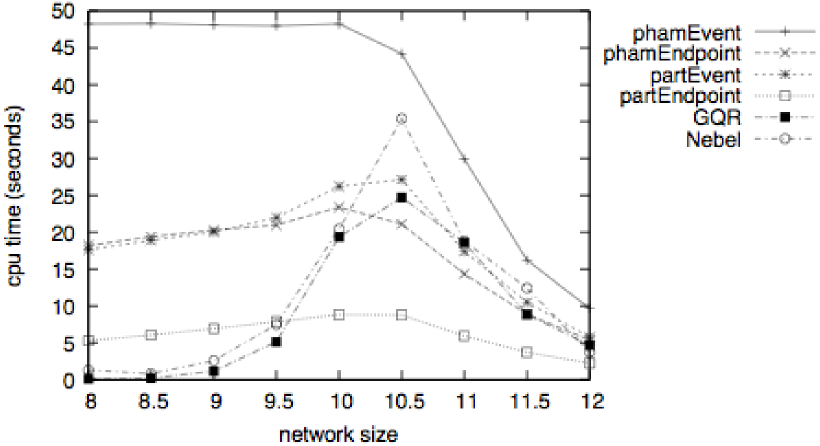
Encode 2 partitions separately

- # of triangles from 20 to 11
- partition recursively
- soundness nontrivial

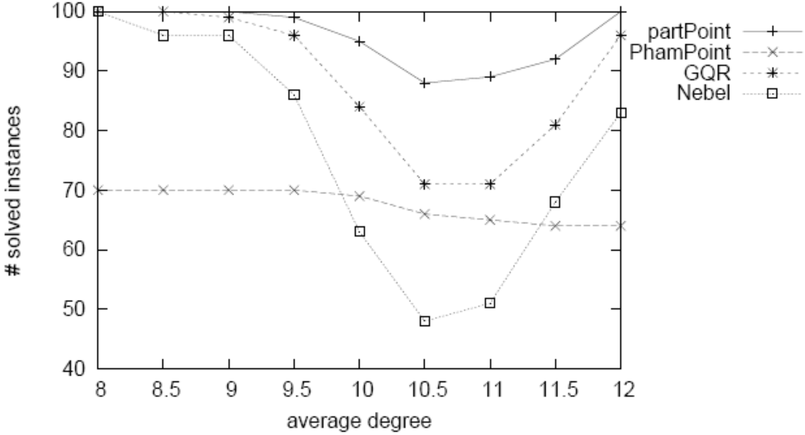
Theorem

IA has atomic network amalgamation property

Empirical Results: 50–100 Nodes



Empirical Results: 110–200 Nodes



Applications of SAT

- Qualitative temporal reasoning
- Constraint solving

A Constraint Model

```
int: z = 10;
array [1..z] of 1..z*z: sq = [x*x | x in 1..z];
array [1..z] of var 0..z: s;
var 1..z: k;
var 1..z: j;
constraint forall ( i in 2..z ) ( s[i] > 0 -> s[i-1] > s[i] );
constraint s[1] < k;
constraint sum ( i in 1..z ) ( sq[s[i]] ) = sq[k];
constraint s[j] > 0;
solve maximize j;
```

Perfect Square: Find largest set of integers $\subseteq \{1, \ldots, z\}$ whose squares sum up to a square

Elements of Constraint Model

- Integer and set comparisons
- Integer arithmetic
- Linear equalities and inequalities
- Set operations
- Array access with variable index
- Global constraints
- Satisfaction and optimization

Existing Methods

- Pseudo-Boolean constraints to SAT
- Boolean cardinality constraints to SAT
- Integer linear constraints to SAT
- Extensional constraints to SAT
- Set constraints to BDDs
- Satisfiability modulo theories
- Lazy clause generation (hybrid of FD and SAT)

New Approach

Everything to SAT

Challenge

Desired encoding varies with constraint type

- Unary suits cardinality constraints
- *Direct encoding* suits extensional constraints
- *Primitive comparisons* can encode linear constraints
- None good for arbitrary arithmetic

Solution

One-size-fits-all binary encoding

- Arbitrary arithmetic supported
- Heterogeneous model into single Boolean formula
- Con: potential loss of propagation power

Adopt constraint language MiniZinc

- Reasonably simple yet expressive
- Many benchmarks and solvers available for empirical study

MiniZinc

- Developed by G12 @ NICTA
- Solver independent modeling
- Reasonable compromise between simplicity & expressivity
- Comes with translation to FlatZinc, suitable as low-level solver input language
- Large pool of benchmarks & examples
- Encourages comparison of different solvers

Binary Encoding

- Use k bits per integer, in two's complement
- Balance between efficiency and completeness
  - Large k: large encoding, inefficient
  - Small k: may fail to find solution
- Start with k sufficient for constants in model
- Increase k, re-encode, and re-solve until solution found or limit (32, e.g.) reached

Integer Comparisons

- $x_3 x_2 x_1 \le y_3 y_2 y_1$

Integer Comparisons

- $x_3 x_2 x_1 \leq y_3 y_2 y_1 \longrightarrow$

  $(x_3 > y_3) \vee [(x_3 = y_3) \wedge (x_2 x_1 \leq_{unsigned} y_2 y_1)]$

Integer Comparisons

- $x_3 x_2 x_1 \leq y_3 y_2 y_1 \longrightarrow$

  $(x_3 > y_3) \vee [(x_3 = y_3) \wedge (x_2 x_1 \leq_{unsigned} y_2 y_1)]$

- $x_2 x_1 \leq_{unsigned} y_2 y_1 \longrightarrow$

  $(x_2 < y_2) \vee [(x_2 = y_2) \wedge (x_1 \leq y_1)]$

Integer Comparisons

- $x_3 x_2 x_1 \leq y_3 y_2 y_1 \longrightarrow$

  $(x_3 > y_3) \vee [(x_3 = y_3) \wedge (x_2 x_1 \leq_{unsigned} y_2 y_1)]$

- $x_2 x_1 \leq_{unsigned} y_2 y_1 \longrightarrow$

  $(x_2 < y_2) \vee [(x_2 = y_2) \wedge (x_1 \leq y_1)]$

- Similar for other operators: $=, \neq, <, \geq, >$

Integer Arithmetic

- Adder and multiplier as in computer hardware
- Constraints to prevent overflow

  $+$: sum temporarily has $k+1$ bits
  leading two bits must be identical

  $\times$: product temporarily has $2k$ bits
  leading $k+1$ bits must be identical

Integer Arithmetic

- Adder and multiplier as in computer hardware
- Constraints to prevent overflow

  $+$: sum temporarily has $k+1$ bits
  leading two bits must be identical

  $\times$: product temporarily has $2k$ bits
  leading $k+1$ bits must be identical
- Subtraction, division, and modulo via $+$ and $\times$

Integer Arithmetic

- Adder and multiplier as in computer hardware
- Constraints to prevent overflow

  $+$: sum temporarily has $k+1$ bits
  leading two bits must be identical

  $\times$: product temporarily has $2k$ bits
  leading $k+1$ bits must be identical
- Subtraction, division, and modulo via $+$ and $\times$
- $max(x, y, z) \longrightarrow$
  $[(y \leq x) \rightarrow (x = z)] \wedge [(y > x) \rightarrow (y = z)]$

Integer Arithmetic

- Adder and multiplier as in computer hardware
- Constraints to prevent overflow

  $+$: sum temporarily has $k+1$ bits
  leading two bits must be identical

  $\times$: product temporarily has $2k$ bits
  leading $k+1$ bits must be identical
- Subtraction, division, and modulo via $+$ and $\times$
- $max(x, y, z) \longrightarrow$
  $[(y \leq x) \rightarrow (x = z)] \wedge [(y > x) \rightarrow (y = z)]$
- Other operators: negation, absolute value, min

Linear Constraints

- $a_1x_1 + \ldots + a_nx_n$ op b
  op can be $=, \neq, \leq, <, \geq, >$

Linear Constraints

- $a_1 x_1 + \ldots + a_n x_n$ op b
  op can be $=, \neq, \leq, <, \geq, >$
- Decompose into multiplications, summations, comparison
- Use auxiliary variables to keep size linear in n

Set Operations

- `set of` $1..10$: Y

Set Operations

- $\texttt{set of } 1..10 : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$

Set Operations

- $\texttt{set of } 1..10 : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$
- Membership: $x \in Y \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge Y_i]$

Set Operations

- $\texttt{set of 1..10} : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$

- Membership: $x \in Y \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge Y_i]$
- Subset: $X \subseteq Y \longrightarrow \bigwedge_{i=1}^{10}(X_i \rightarrow Y_i)$

Set Operations

- `set of` $1..10 : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$

- Membership: $x \in Y \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge Y_i]$
- Subset: $X \subseteq Y \longrightarrow \bigwedge_{i=1}^{10}(X_i \rightarrow Y_i)$
  - If X and Y have different universes, use smallest range containing both

Set Operations

- $\texttt{set of 1..10} : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$

- Membership: $x \in Y \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge Y_i]$
- Subset: $X \subseteq Y \longrightarrow \bigwedge_{i=1}^{10}(X_i \rightarrow Y_i)$
  - If X and Y have different universes, use smallest range containing both
- Cardinality: $x = |Y| \longrightarrow x = \sum_{i=1}^{10} Y_i$

Set Operations

- $\texttt{set of } 1..10 : Y \longrightarrow Y_1, \ldots, Y_{10}$
- Y_i encodes $i \in Y$

- Membership: $x \in Y \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge Y_i]$
- Subset: $X \subseteq Y \longrightarrow \bigwedge_{i=1}^{10}(X_i \rightarrow Y_i)$
  - If X and Y have different universes, use smallest range containing both
- Cardinality: $x = |Y| \longrightarrow x = \sum_{i=1}^{10} Y_i$
- Similar for other operators: union, intersection, difference, symmetric difference

Arrays of Booleans/Integers/Sets

- Index range fixed at compile time
- Decompose into individual variables
  $Y[1..10] \longrightarrow Y_1, \ldots, Y_{10}$
- Handling variable indices
  $Y[x] = z \longrightarrow \bigvee_{i=1}^{10}[(x = i) \wedge (Y_i = z)]$

Optimization Problems

- Optimization of variable only (optimization of expression eliminated using auxiliary variable)
- Binary search for increasingly optimal solutions
- Each step of search is a satisfaction problem
- At most $k+1$ subproblems (log of domain size)

Complexity of Encoding

- Quadratic in *k* for $\times, /, \%$, linear constraints
- Linear for $+, -, ||, min, max, =, \neq, \leq, <, \geq, >$
- Linear in size of array for array access with variable index
- Linear in size of universe of set for set constraints
- In practice, often millions of variables & clauses

Weaknesses and Strengths

- Domain knowledge lost
- Binary search blind
- Propagation weak for some types of constraints

- All constraints propagated seamlessly at once
- Clause learning more powerful than traditional nogood learning
- SAT heuristics good at real-world problems

Empirical Evaluation

- Use all benchmark groups & examples in MiniZinc distribution (3/3/2008)
- 488 problems in 21 groups: 12 satisfaction, 8 optimization, 1 mixed
  - rectangle packing, linear equations, car sequencing, curriculum design, social golfers, job shop scheduling, nurse scheduling, n-queens, truck scheduling, warehouse planning, math puzzles, ...
- Compare with G12/FD & Gecode/FlatZinc
- 4-hour time limit for each run

Overall Result

\# of problems solved out of 488

FznTini	G12/FD	Gecode/FlatZinc
263	103	178

Easy Cases for All

Problem	Inst.	FznTini		G12/FD		Gecode/FlatZinc	
		Solved	Time	Solved	Time	Solved	Time
alpha	1	1	1.65	1	0.10	1	239.20
areas	4	4	0.69	4	0.71	4	0.04
eq	1	1	49.92	1	0.18	1	0.00
examples	18	18	2076.74	18	1557.62	18	2.87
kakuro	6	6	0.17	6	1.10	6	0.01
knights	4	4	0.78	4	390.79	4	1.01
photo	1	1	0.08	1	0.20	1	0.00

Bad Cases for Booleanization

Problem	Inst.	FznTini		G12/FD		Gecode/FlatZinc	
		Solved	Time	Solved	Time	Solved	Time
cars	79	1	3.34	1	0.15	1	0.01
golfers	9	3	6278.30	4	12.88	6	1297.26
golomb	5	4	2030.23	5	323.54	5	10.35
magicseq	7	4	9939.32	7	172.12	7	9.19
queens	6	5	4168.79	6	90.68	3	0.33
trucking	10	9	14747.10	10	196.48	10	86.52

Good Cases for Booleanization

Problem	Inst.	FznTini		G12/FD		Gecode/FlatZinc	
		Solved	Time	Solved	Time	Solved	Time
packing	50	9	2843.46	7	2447.65	7	44.13
bibd	9	8	16.17	8	757.72	6	197.48
curriculum	3	3	12.76	2	13.17	0	—
jobshop	73	19	50294.40	2	1764.65	2	31.6
nurse sch.	100	99	1800.36	1	3.97	0	—
perfect sq.	10	10	548.41	4	4350.19	5	2024.85
warehouses	10	10	671.71	9	2266.44	9	221.83

Summary and Opportunities

- SAT works!
- More compact and/or propagation friendly encodings of constraints
- Direct encoding of global constraints
- More informed search (than blind binary search) for optimization problems
- Deeper empirical and theoretical studies of power and limitations
- Hybridizations of SAT and other techniques

Extensions of SAT

- Max-SAT
- Model counting
- Knowledge compilation
- Quantified Boolean formulas
- Pseudo-Boolean formulas

Max-SAT

Satisfy max # of clauses

Clauses can have weights

- satisfy clauses with max sum of weights

Can have *hard* clauses

- these must be satisfied
- maximize with respect to rest

Model Counting

Compute # of models (satisfying assignments)

$\#P$-complete

Literals can have weights

- weight of model: product of literal weights
- compute sum of weights of models
- closely related to probabilistic reasoning

Knowledge Compilation

Put formula into given logical form

- to support efficient (poly-time) operations

Target compilation forms

- decomposable negation normal form
- binary decision diagrams
- prime implicates, ...

Forms differ in succinctness & tractability

- pick most succinct form that supports desired operations
- need to develop compilers for them

Quantified Boolean Formula

$$\forall X \exists Y \forall Z \, (X \vee Z) \wedge Y$$

Quantified Boolean Formula

$\forall X \exists Y \forall Z \, (X \lor Z) \land Y$

- all variables quantified

Is formula true?

PSPACE-complete

Pseudo-Boolean Formula

Pseudo-Boolean constraint

- $2X + Y + 3Z < 5$

Pseudo-Boolean Formula

Pseudo-Boolean constraint

- $2X + Y + 3Z < 5$

Special case: cardinality constraint

- $X + Y + Z > 2$

SAT Resources

- SAT conferences, www.satisfiability.org
- SAT competitions, www.satcompetition.org
- SAT Live, www.satlive.org
- Handbook of satisfiability