Algorithm for SAT
- MiniSAT Heuristics

Moonzoo Kim

KAIST Korea Advanced Institute of
Science and Technology

MiniSAT SAT Solver

- Overview
- Conflict Clause Analysis
- VSIDS Decision Heuristic

SAT Solver History

- Started with DPLL (1962)
  - Able to solve 10-15 variable problems
- Satz (Chu Min Li, 1995)
  - Able to solve some 1000 variable problems
- Chaff (Malik et al., 2001)
  - Intelligently hacked DPLL , Won the 2004 competition
  - Able to solve some 10000 variable problems
- Current state-of-the-art
  - MiniSAT and SATELITEGTI (Chalmer's university, 2004-2006)
  - Jerusat and Haifasat (Intel Haifa, 2002)
  - Ace (UCLA, 2004-2006)

Overview

- MiniSat is a fast SAT solver developed by Niklas Eén and Niklas Sörensson
  - MiniSat won all industrial categories in SAT 2005 competition
  - MiniSat won SAT-Race 2006
- MiniSat is simple and well-documented
  - Well-defined interface for general use
  - Helpful implementation documents and comments
  - Minimal but efficient heuristic

Overview

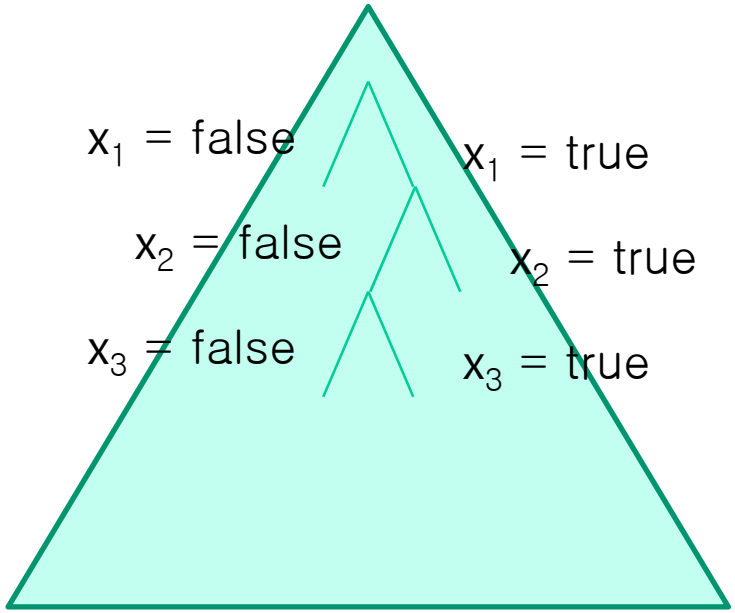
- Unit clause is a clause in which all but one of literals is assigned to False
- Unit literal is the unassigned literal in a unit clause

$$\ldots\ldots$$
$$(x_0) \wedge$$
$$(-x_0 \vee x_1) \wedge$$
$$(-x_2 \vee -x_3 \vee -x_4) \wedge$$
$$\ldots\ldots$$

  - (x_0) is a unit clause and 'x_0' is a unit literal
  - $(-x_0 \vee x_1)$ is a unit clause since x_0 has to be True
  - $(-x_2 \vee -x_3 \vee -x_4)$ can be a unit clause if the current assignment is that x_3 and x_4 are True
- Boolean Constrain Propagation(BCP) is the process of assigning the True value to all unit literals

Overview

```
/* overall structure of Minisat solve
    procedure */
Solve(){
    while(true){
        boolean_constraint_propagation();
        if(no_conflict){
            if(no_unassigned_variable) return SAT;
            decision_level++;
            make_decision();
        }else{
            if (no_dicisions_made) return UNSAT;
            analyze_conflict();
            undo_assignments();
            add_conflict_clause();
        }
    }
}
```



Conflict Clause Analysis

- A conflict happens when one clause is falsified by unit propagation

Assume x_4 is False
$(x_1 \vee x_4) \wedge$
$(-x_1 \vee x_2) \wedge$
$(-x_2 \vee x_3) \wedge$
$(-x_3 \vee -x_2 \vee -x_1)$ Falsified!

- Analyze the conflicting clause to infer a clause
  - $(-x_3 \vee -x_2 \vee -x_1)$ is conflicting clause
- The inferred clause is a new knowledge
  - A new learnt clause is added to constraints

Conflict Clause Analysis

- Learnt clauses are inferred by conflict analysis

$(x_1 \vee x_4) \wedge$
$(-x_1 \vee x_2) \wedge$
$(-x_2 \vee x_3) \wedge$
$(-x_3 \vee -x_2 \vee -x_1) \wedge$
(x_4) **learnt clause**

- They help prune future parts of the search space
  - Assigning False to x_4 is the casual of conflict
  - Adding (x4) to constraints prohibit conflict from $-x_4$
- Learnt clauses actually drive backtracking

Conflict Clause Analysis

```
/* conflict analysis algorithm */
Analyze_conflict(){
    cl = find_confclicting_clause();
    /* Loop until cl is falsified and one literal whose value is determined in current
    decision level is remained */
    While(!stop_criterion_met(cl)){
        lit = choose_literal(cl); /* select the last propagated literal */
        Var = variable_of_literal(lit);
        ante = antecedent(var);
        cl = resolve(cl, ante, var);
    }
    add_clause_to_database(cl);
    /* backtrack level is the lowest decision level for which the learnt clause is unit
    clause */
    back_dl = clause_asserting_level(cl);
    return back_dl;
}
```

Algorithm from Lintao Zhang and Sharad malik
"The Quest for Efficient Boolean Satisfiability Solvers"

Conflict Clause Analysis

- Example of conflict clause analysis

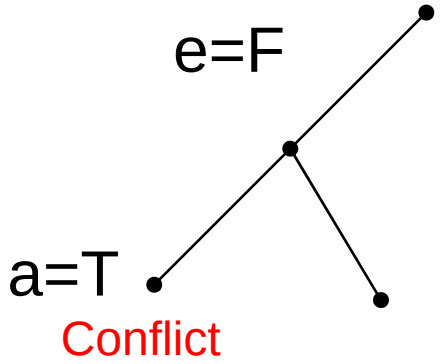
$$(-f \vee e) \wedge$$
$$(-g \vee f) \wedge$$
$$(b \vee a \vee e) \wedge$$
$$(c \vee e \vee f \vee -b) \wedge$$
$$(d \vee -b \vee h) \wedge$$
$$(-b \vee -c \vee -d) \wedge$$
$$(c \vee d)$$

Satisfiable?

Unsatisfiable?

Conflict Clause Analysis

Assignments		antecedent
e=F		assumption
f=F		-f ∨ e
g=F	DLevel=1	-g ∨ f
h=F		-h ∨ g
a=F		assumption
b=T	DLevel=	b ∨ a ∨ e
c=T	2	c ∨ e ∨ f ∨ -b
d=T		d ∨ -b ∨ h

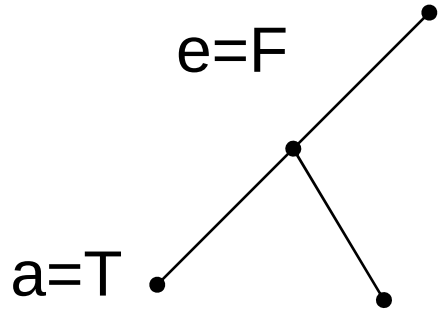


-b ∨ -c ∨ -d

Example slides are from CMU 15-414 course ppt

Conflict Clause Analysis

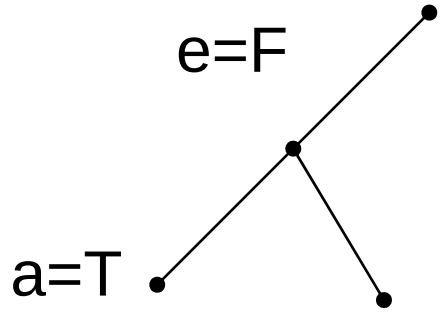
Assignments		antecedent
e=F		assumption
f=F	DLevel=1	-f ∨ e
g=F		-g ∨ f
h=F		-h ∨ g
a=F		assumption
b=T		b ∨ a ∨ e
c=T	DLevel=2	c ∨ e ∨ f ∨ -b
d=T		d ∨ -b ∨ h



-b ∨ -c ∨ -d

Conflict Clause Analysis

Assignments		antecedent
e=F		assumption
f=F	DLevel=1	-f ∨ e
g=F		-g ∨ f
h=F		-h ∨ g
a=F		assumption
b=T		b ∨ a ∨ e
c=T	DLevel= 2	c ∨ e ∨ f ∨ -b
d=T		d ∨ -b ∨ h

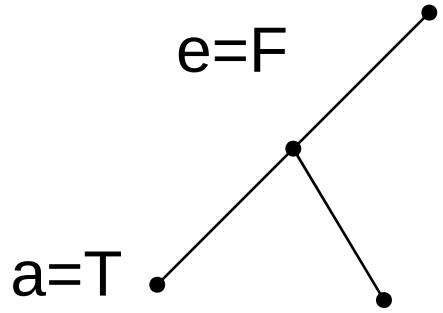


-b ∨ -c ∨ -d

-b ∨ -c ∨ h

Conflict Clause Analysis

Assignments		antecedent
e=F		assumption
f=F	DLevel=1	-f ∨ e
g=F		-g ∨ f
h=F		-h ∨ g
a=F		assumption
b=T		b ∨ a ∨ e
c=T	DLevel= 2	c ∨ e ∨ f ∨ -b
d=T		d ∨ -b ∨ h

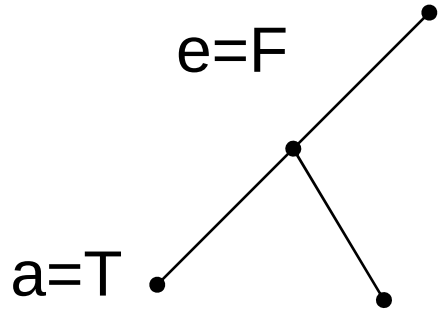


-b ∨ -c ∨ -d

-b ∨ -c ∨ h

Conflict Clause Analysis

Assignments		antecedent
e=F		assumption
f=F	DLevel=1	-f ∨ e
g=F		-g ∨ f
h=F		-h ∨ g
a=F		assumption
b=T	DLevel= 2	b ∨ a ∨ e
c=T		c ∨ e ∨ f ∨ -b
d=T		d ∨ -b ∨ h



-b ∨ -c ∨ -d

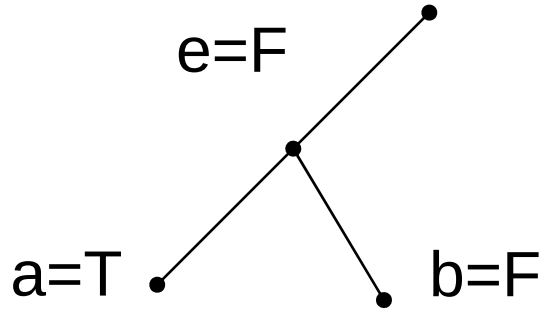
-b ∨ -c ∨ h

-b ∨ e ∨ f ∨ h learnt clause

Conflict Clause Analysis

Assignments	antecedent
e=F	assumption
f=F	-f ∨ e
g=F	-g ∨ f
h=F	-h ∨ g
b=F	-b ∨ e ∨ f ∨ h
…	…

DLevel=1



-b ∨ -c ∨ -d

-b ∨ -c ∨ h

-b ∨ e ∨ f ∨ h

VSIDS Decision Heuristic

- Variable State Independent Decaying Sum(VSIDS)
  - decision heuristic to determine what variable will be assigned next
  - decision is independent from the current assignment of each variable
- VSIDS makes decisions based on activity
  - Activity is a literal occurrence count with higher weight on the more recently added clauses
  - MiniSAT does not consider any polarity in VSIDS
    - Each variable, not literal has score

activity description from Lintao Zhang and Sharad malik
“The Quest for Efficient Boolean Satisfiability Solvers”

VSIDS Decision Heuristic

- Initially, the score for each variable is 0
- First make a decision e = False
  - The order between same score is unspecified.
  - MiniSAT always assigns False to variables.

Initial constraints
(-f∨e) ∧
(-g∨f) ∧
(b∨a∨e) ∧
(c∨e∨f∨-b) ∧
(d∨-b∨h) ∧
(-b∨-c∨-d) ∧
(c∨d)

Variable	Score	Value
a	**0**	
b	**0**	
c	**0**	
d	**0**	
e	**0**	**F**
f	**0**	
g	**0**	
h	**0**	

VSIDS Decision Heuristic

- f, g, h are False after BCP

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d)

Variable	Score	Value
a	**0**	
b	**0**	
c	**0**	
d	**0**	
e	**0**	**F**
f	**0**	**F**
g	**0**	**F**
h	**0**	**F**

VSIDS Decision Heuristic

- a is next decision variable

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d)

Variable	Score	Value
a	**0**	**F**
b	**0**	
c	**0**	
d	**0**	
e	**0**	**F**
f	**0**	**F**
g	**0**	**F**
h	**0**	**F**

VSIDS Decision Heuristic

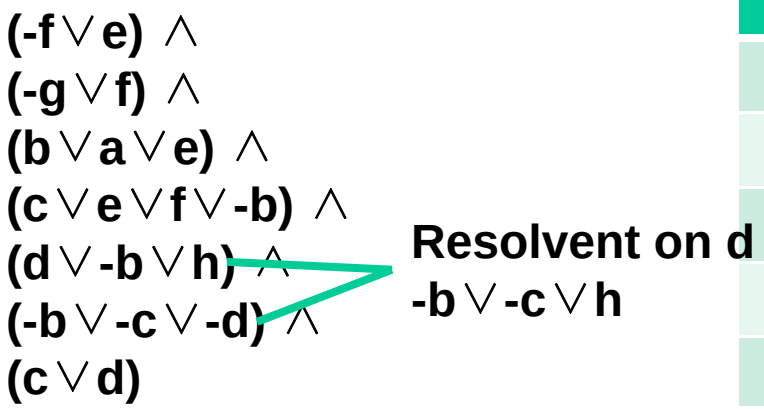
- b, c are True after BCP
- Conflict occurs on variable d
  - Start conflict analysis

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d)

Variable	Score	Value
a	0	F
b	0	T
c	0	T
d	0	T
e	0	F
f	0	F
g	0	F
h	0	F

VSIDS Decision Heuristic

- The score of variable in resolvents is increased by 1
- If a variable appears in resolvents two or mores increase the score just once



Variable	Score	Value
a	0	F
b	1	T
c	1	T
d	0	T
e	0	F
f	0	F
g	0	F
h	1	F

VSIDS Decision Heuristic

- The end of conflict analysis
- The scores are decaying 5% for next scoring

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d)

Resolvents
-b ∨ -c ∨ h
-b ∨ e ∨ f ∨ h ← learnt clause

Variable	Score	Value
a	0	F
b	0.95	T
c	0.95	T
d	0	T
e	0.95	F
f	0.95	F
g	0	F
h	0.95	F

VSIDS Decision Heuristic

- b is now False and a is True after BCP
- Next decision variable is c with 0.95 score

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d) ∧
Learnt clause (-b ∨ e ∨ f ∨ h)

Variable	Score	Value
a	**0**	**T**
b	**0.95**	**F**
c	**0.95**	
d	**0**	
e	**0.95**	**F**
f	**0.95**	**F**
g	**0**	**F**
h	**0.95**	**F**

VSIDS Decision Heuristic

- Finally we find a model

(-f ∨ e) ∧
(-g ∨ f) ∧
(b ∨ a ∨ e) ∧
(c ∨ e ∨ f ∨ -b) ∧
(d ∨ -b ∨ h) ∧
(-b ∨ -c ∨ -d) ∧
(c ∨ d) ∧
Learnt clause (-b ∨ e ∨ f ∨ h)

Variable	Score	Value
a	**0**	**T**
b	**0.95**	**F**
c	**0.95**	**F**
d	**0**	**T**
e	**0.95**	**F**
f	**0.95**	**F**
g	**0**	**F**
h	**0.95**	**F**

Basic References

- The Quest for Efficient boolean Satisfiability Solvers

  L. Zhang and S. Malik

  Computer Aided Verification, ***Denmark, July 2002 (LNCS 2404)***
- A tool for checking ANSI-C programs

  E. Clarke, D. Kroening and F. Lerda

  Tools and Algorithms for the Construction and Analysis of Systems, ***Spain, 2004 (LNCS 2988)***
- Backdoors To Typical Case Complexity.

  R. Williams, C. Gomes, and B. Selman.

  International Joint Conference on Artificial Intelligence