Modern SAT Solving

Johannes Fichte

TDDD34 (Guest Lecture)

Linköping University
April 23, 2026

SAT Problem

Boolean Satisfiability Problem

SAT-Problem (Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Input normal form

- Conjunctive normal form (CNF)
- Form:

  $F = (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \ldots \wedge (\ldots)$ where ℓ_i either x or $\neg x$
- We can transform efficiently (polynomial-time) to CNF (Tseitin Transformation).

Boolean Satisfiability Problem

SAT-Problem (Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Input normal form

- Conjunctive normal form (CNF)
- Form:

  $F = (\ell_1 \vee \ell_2 \vee \ell_3) \wedge \ldots \wedge (\ldots)$ where ℓ_i either x or $\neg x$
- We can transform efficiently (polynomial-time) to CNF (Tseitin Transformation).

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

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

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

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

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

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

$\tau = \{a \mapsto 1\}$

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.

Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

$$(a \vee \neg b \vee d) \overset{1}{\wedge} (\neg a \vee \neg c \vee \neg d) \overset{1}{\wedge} (\neg a \vee \neg d) \wedge (b \vee c) \wedge (d \vee e) \wedge (\neg b \vee \neg e)$$

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.

Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

$$(a \vee \neg b \vee d) \overset{1}{\wedge} (\neg a \vee \neg c \vee \neg d) \overset{1}{\wedge} (\neg a \vee \neg d) \overset{1}{\wedge} (b \vee c) \wedge (d \vee e) \wedge (\neg b \vee \neg e)$$

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.

Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.

Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

Boolean Satisfiability Problem

SAT-Problem

(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.

Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

Boolean Satisfiability Problem

SAT-Problem

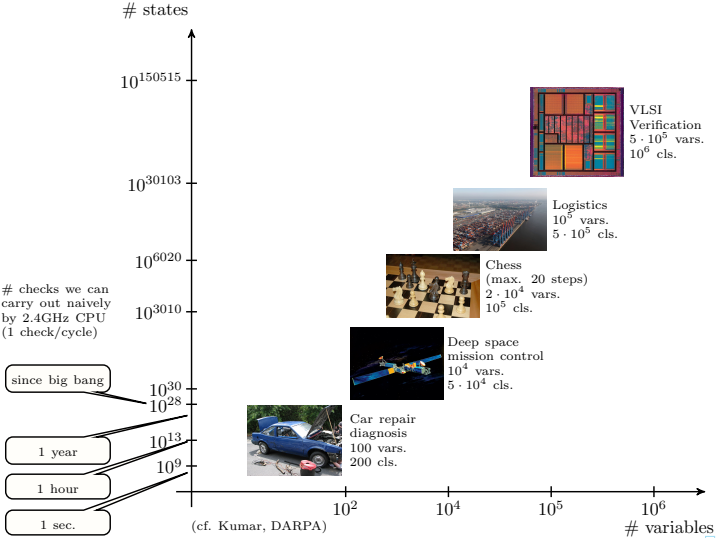
(Fichte, Berre, Hecher, and Szeider, 2023)

Given: Propositional formula F.
Question: Is there an assignment τ to $var(F)$ such that F_τ evaluates to 1 (*satisfiable*).

Example

$\Rightarrow$ Satisfiable

Practical Challenges



Very short History of SAT solving

1960's Early techniques: DPLL
by Davis & Putnam (1962) and Davis & Logemann & Loveland (1962)

1970's Satisfiability is fundamental problem of mathematics
We can blame it on Cook (1971) and Levin (1973)

1990's Can we solve SAT in practice?

2000's SAT Revolution

2010's Widely applied. Open: When and why?

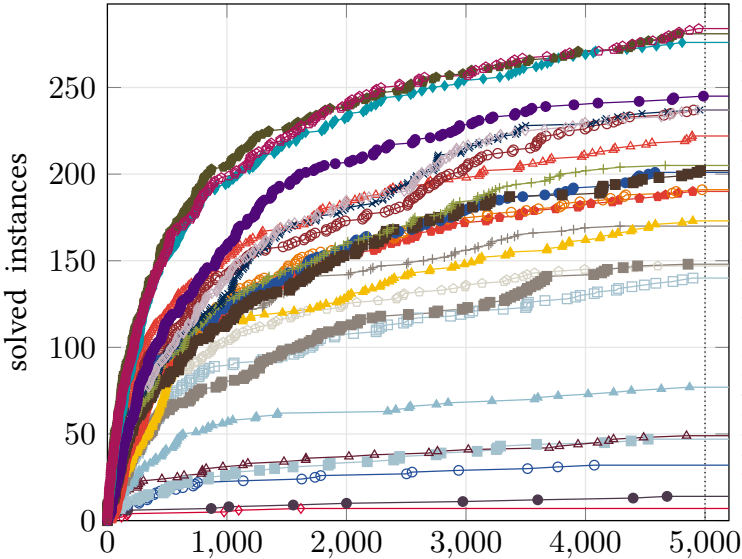
2020's Silent (R)evolution: widely applied.

Practical Challenges

Competition

- 1991: 1st competition
  - Max. 215 variables
- Nowadays
  - Up to $15,000,000$ - $20,000,000$ variables
  - Automated theorem proving
  - Hardware and software verification
  - SAT embedded in complex procedures

SAT Competition All Time Winners on SAT Competition 2022 Benchmarks



Practical Challenges: Algorithms vs Hardware

Fun Experiment

20-year-old SAT-solvers

Modern SAT-solvers

- Timespan: two decades
- Design virtual teams and let them compete

Which team can solve more instances?

Challenges

What are the reasons for such tremendous improvements?

Possible answers:

- Sophisticated implementations
- Better clause learning
- Non-chronological backtracking
- Diverse Heuristics, Solver restarts

Practical Implementations

Competition

- 1991: 1st competition
  - Max. 215 vars.
- 2014: latest competition
  - Max. $2,335,191$ vars. (combinatorial hard track)
  - Max. $13,842,706$ vars. (application track)

In Contrast

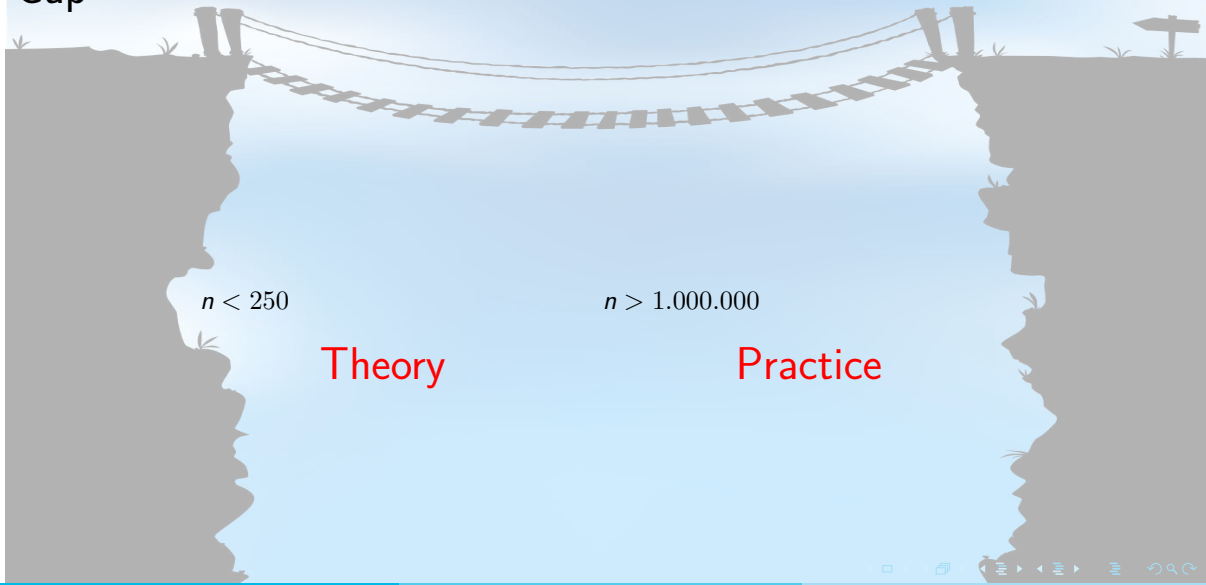
Theoretical (worst case) bounds

2^n	trivial
1.333^n	1999 (Schoening)
1.3302^n	2002
1.3290^n	2003
1.3280^n	2003
1.324^n	2010 (Hertli)

Reminder

For n=250 that exceeds the number of nano seconds that passed since the big bang!

Gap

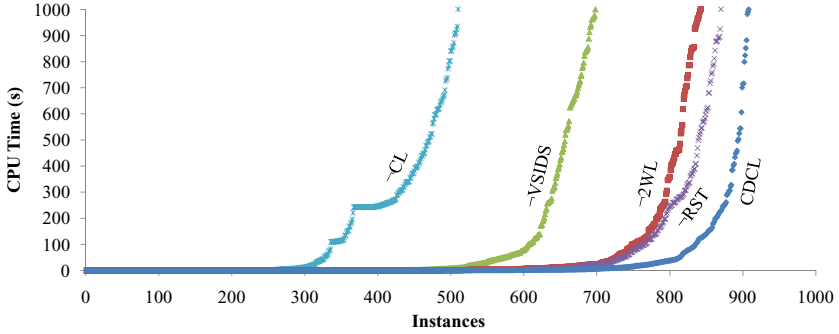


What makes solvers so fast? (in practice)

- Conflict Driven Clause Learning (CDCL)
- Restarting and Forgetting
- Dynamic Branching Heuristics (VSIDS)
- Pre/Inprocessing
  - Variables
    - Unit Propagation (UP)
    - Pure Literals, Equivalent literals, Failed literals
    - Tautologies, Subsumed clauses, Self-subsuming resolution
    - Bounded variable elimination
  - Clauses: Blocked clauses
  - Binary clauses
    - Transitive Reductions
    - Hidden tautology elimination
  - Adding Redundancy
    - Hyper binary resolution
    - Bounded variable addition
    - Clause addition

Empirical Study on Modern Techniques

(Katebi et al., 2011)

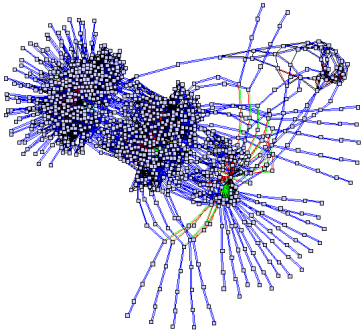


What makes solving so fast? (in theory)

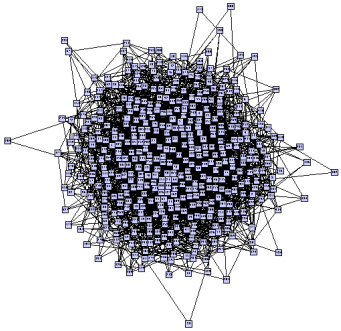
- We don't fully know. But we have some indicators.
- Structure in the instance matters. ⇒ Different Course

Idea: Structure Matters

Exploit the existence of a “hidden structure” (parameter)



Industrial Instance
(multiplier verification)



Random Instance

Basic Solving Techniques

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

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

$(d \vee e) \wedge (\neg d \vee e) \wedge (\neg b \vee \neg e)$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

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

$(d \vee e) \wedge (\neg d \vee e) \wedge (\neg b \vee \neg e)$

Resolved Clauses:

$$\frac{d \vee e, \quad \neg d \vee e}{e}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

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

$(d \vee e) \wedge (\neg d \vee e) \wedge (\neg b \vee \neg e)$

Resolved Clauses:

e

$$\frac{d \vee e, \quad \neg d \vee e}{e}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolved Clauses:

e

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

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

$(d \vee e) \wedge (\neg d \vee e) \wedge (\neg b \vee \neg e)$

Resolved Clauses:

e

$$\frac{b \vee \neg e, \quad \neg b \vee \neg e}{\neg e}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolved Clauses:

e

$\neg e$

$$\frac{b \vee \neg e, \quad \neg b \vee \neg e}{\neg e}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

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

$(d \vee e) \wedge (\neg d \vee e) \wedge (\neg b \vee \neg e)$

Resolved Clauses:

e

$\neg e$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolved Clauses:

e

$\neg e$

$$\frac{e, \quad \neg e}{\bot}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolved Clauses:

e

$\neg e$

$$\frac{e, \quad \neg e}{\bot}$$

Resolution

(Davis and Putnam, 1960)

Resolution Rule

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \mathbf{x}, \quad \ell'_1 \vee \ldots \vee \ell'_m \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_k \vee \ell'_1 \vee \ldots \vee \ell'_m}$$

Example

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

Resolved Clauses:

e

$\neg e$

$$\frac{e, \quad \neg e}{\bot}$$

$\Rightarrow$ Unsatisfiable

DPLL Algorithm

(Davis et al., 1962)

Idea

- Resolution might need exponential space
- Replace resolution rule by splitting rule (branching)

Problem: resolution might need exponential spaceReplace resolution rule by splitting rule (branching)

Algorithm 1: DPLL(F, τ)

while *True* **do**
  Choose $x \in var(F_\tau), a \in \{0, 1\}$ `/* Heuristic */`
  if *Nothing to Decide* **then**
    return SAT
  if F_τ *contains dissatisfied clause* **then**
    if *Conflict on top level* **then**
      return UNSAT
    Backtrack `/* unassign recent literal */`
    Flip `/* assign complement (` $x \mapsto 1 - a$ `) */`

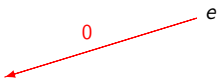
DPLL Algorithm (contd.)

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

e

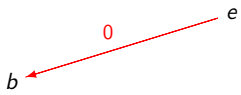
DPLL Algorithm (contd.)

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



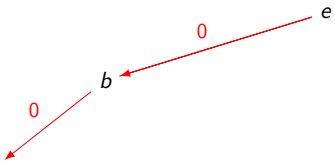
DPLL Algorithm (contd.)

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



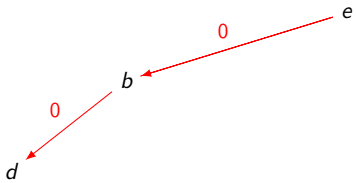
DPLL Algorithm (contd.)

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



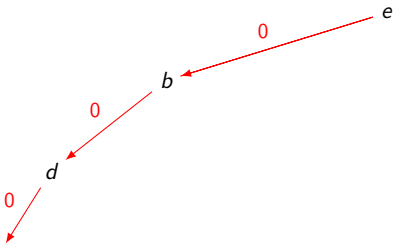
DPLL Algorithm (contd.)

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



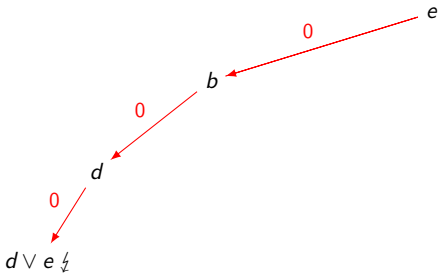
DPLL Algorithm (contd.)

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



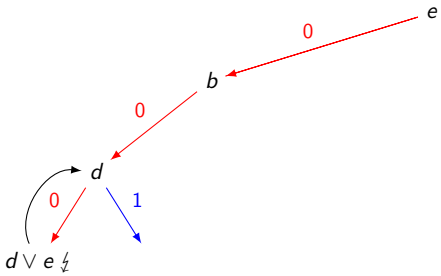
DPLL Algorithm (contd.)

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



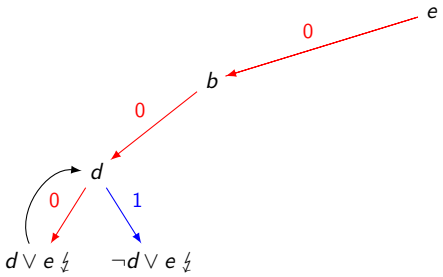
DPLL Algorithm (contd.)

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



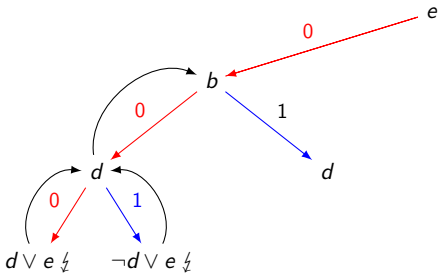
DPLL Algorithm (contd.)

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



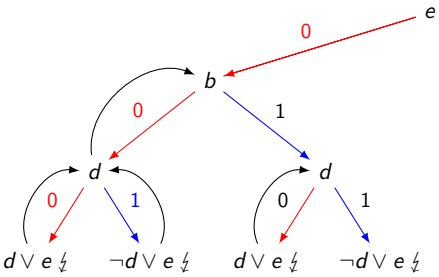
DPLL Algorithm (contd.)

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



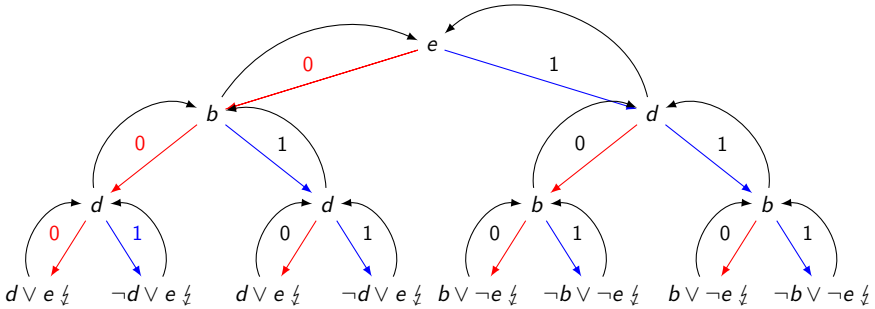
DPLL Algorithm (contd.)

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



DPLL Algorithm (contd.)

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



Simplification Rules

Unit Propagation (UP) (Dowling and Gallier, 1984)

$$\frac{\ell \vee \ell_1 \vee \ldots \vee \ell_m}{\ell}$$

Algorithm 2: DPLL(F, τ)

while *True* **do**
  F := Simplify(F_τ)
  Choose $x \in var(F_\tau), a \in \{0, 1\}$ /* Heuristic */
  if *Nothing to Decide* **then**
    return SAT
  if F_τ *contains dissatisfied clause* **then**
    if *Conflict on top level* **then**
      return UNSAT
    Backtrack /* unassign recent literal */ /* undo simplification */
    Flip /* assign complement ($x \mapsto 1 - a$) */

Simplification Rules (contd.)

Further linear quadratic simplification rules

- Pure Literal
- Tautology
- Horn

Modern SAT Technology (CDCL)

Conflict Driven Clause Learning (CDCL)

Situation

- DPLL runs into conflict, then backtracking starts
- What if same conflicts in a different branch?

$\Rightarrow$ Conflict again, but might need additional assignments

Idea

- Detect conflicts that occurred earlier
- Do not run again in the same conflict
- Learn from conflict and add conflict clause
- Introduced in solver grasp (Silva and Sakallah, 1996)

Conflict Driven Clause Learning (CDCL)

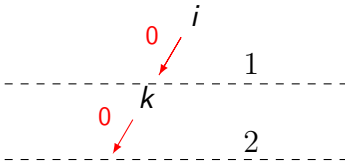
$$
\begin{aligned}
&(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge \\
&(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge \\
&(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge \\
&(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)
\end{aligned}
$$

Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

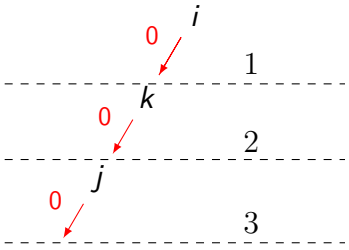
Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



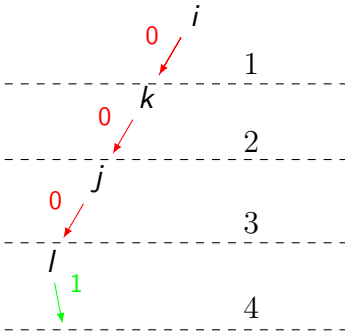
Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



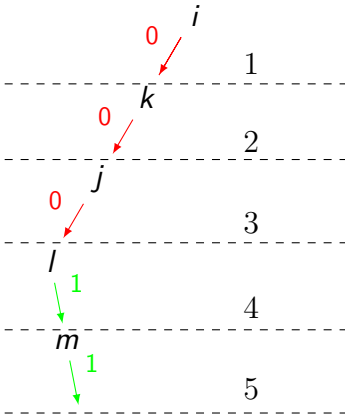
Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



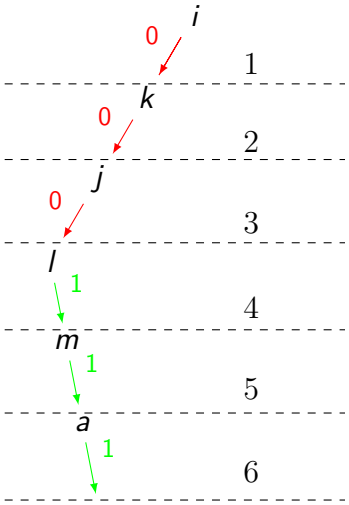
Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



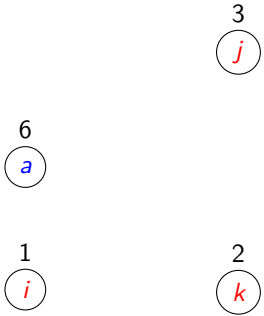
Conflict Driven Clause Learning (CDCL)

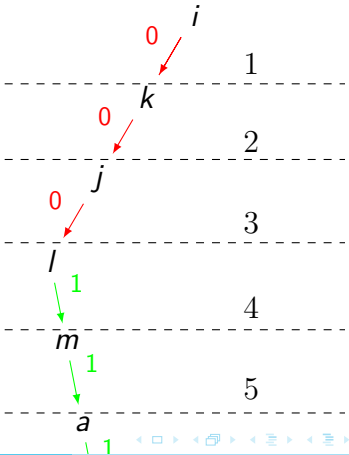
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



Conflict Driven Clause Learning (CDCL)

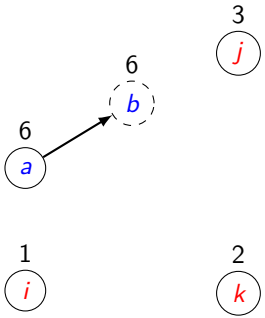
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

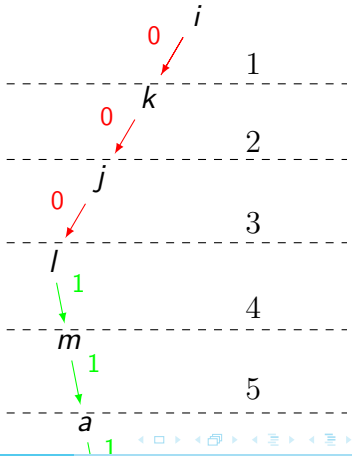




Conflict Driven Clause Learning (CDCL)

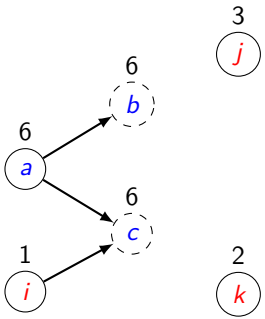
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

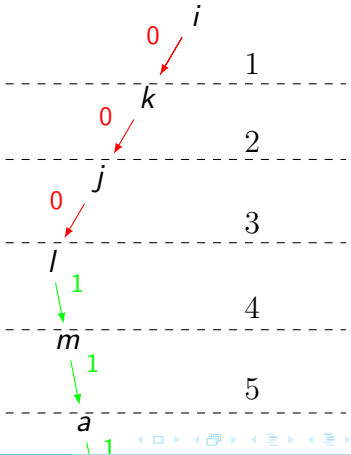




Conflict Driven Clause Learning (CDCL)

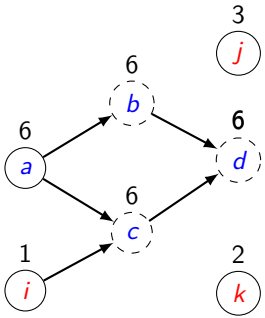
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

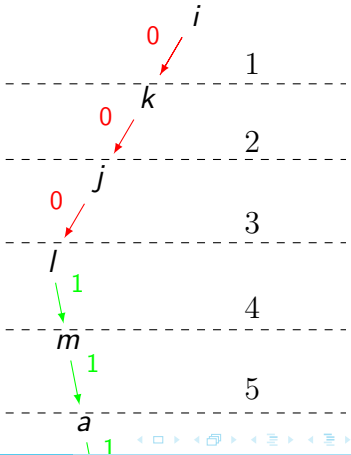




Conflict Driven Clause Learning (CDCL)

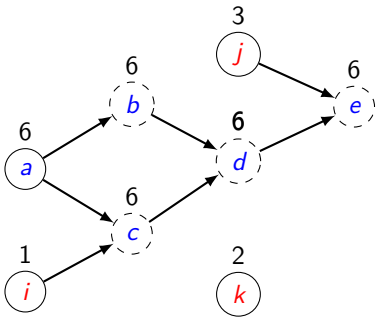
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

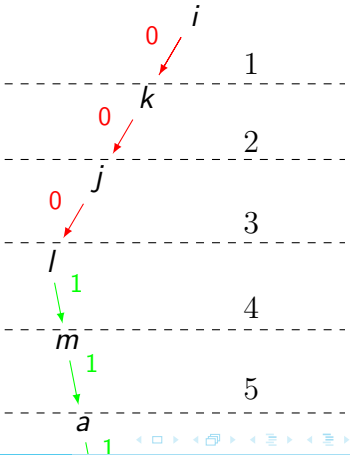




Conflict Driven Clause Learning (CDCL)

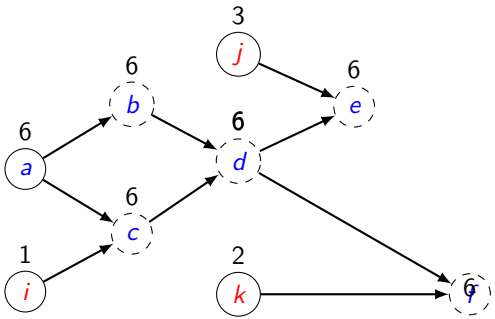
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

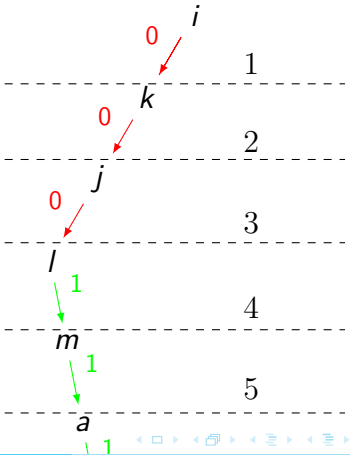




Conflict Driven Clause Learning (CDCL)

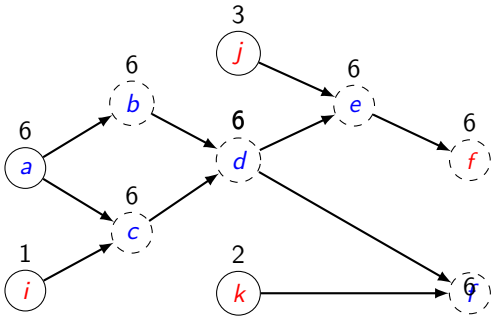
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

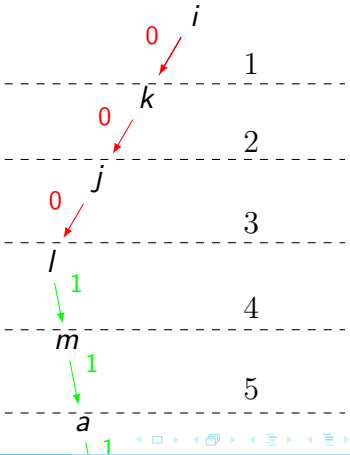




Conflict Driven Clause Learning (CDCL)

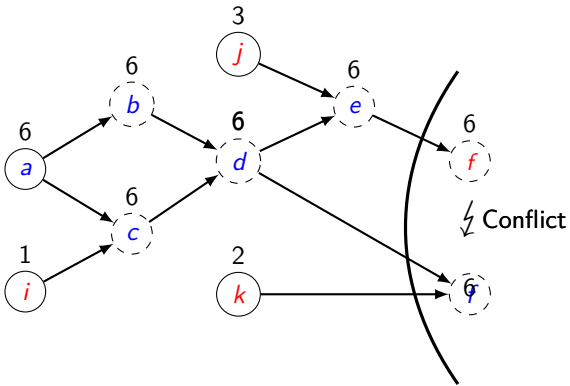
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

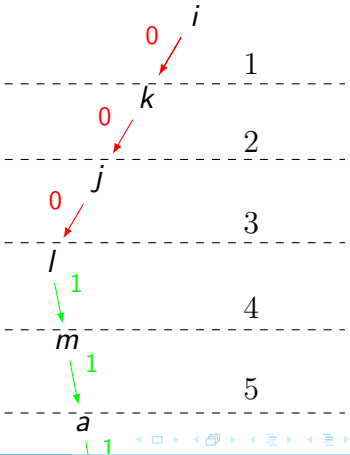




Conflict Driven Clause Learning (CDCL)

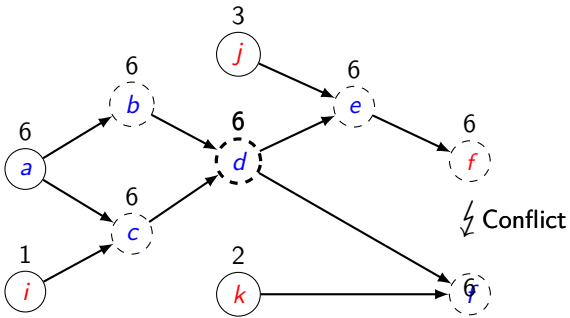
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



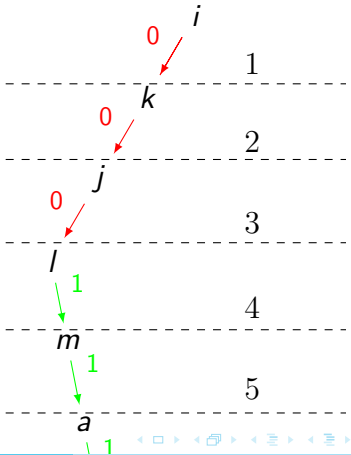


Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

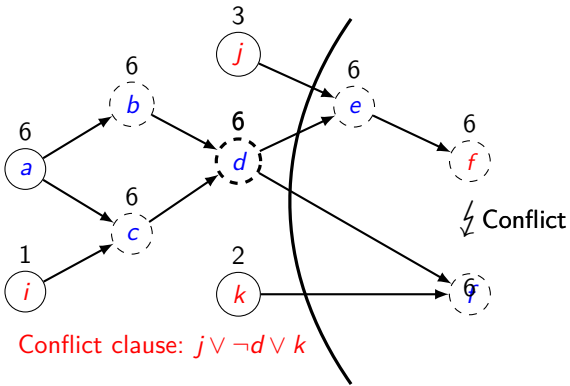


(1st) UIP

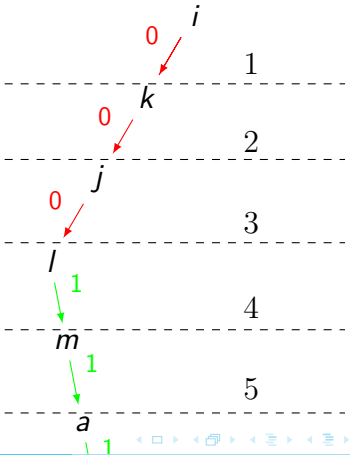


Conflict Driven Clause Learning (CDCL)

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$



Conflict clause: $j \vee \neg d \vee k$



Conflict Driven Clause Learning (CDCL)

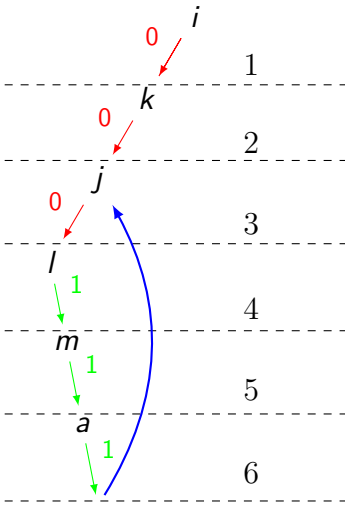
Algorithm 3: CDCL(F, τ)

```
while True do
    F := Simplify(F_τ)                                   /* UP */
    Choose x ∈ var(F_τ), a ∈ {0,1}                       /* Heuristic */
    if Nothing to Decide then
        return SAT
    while F contains dissatisfied clause C_dissat do
        C_confl := CompConflictCls(C_dissat)             /* 1st UIP */
        if C_confl = ⊥ then
            return UNSAT
        Learn(C_conflict)
        Backtrack(C_conflict)
        F := Simplify(F_τ)                               /* UP */
```

Backjumping

$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

No chronological backtracking



Backjumping

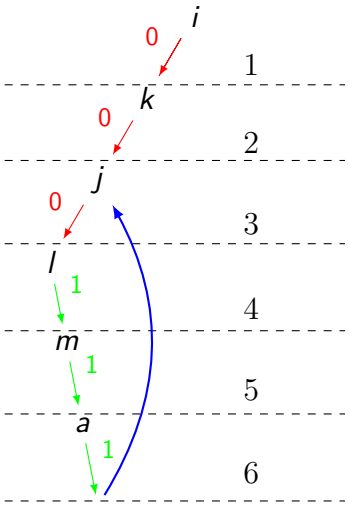
$$(\neg a \vee b) \wedge (\neg a \vee c \vee i) \wedge$$
$$(\neg b \vee \neg c \vee d) \wedge (\neg d \vee e \vee j) \wedge$$
$$(\neg d \vee f \vee k) \wedge (\neg e \vee \neg f) \wedge$$
$$(a \vee g \vee l) \wedge (a \vee h) \wedge (\neg g \vee \neg h \vee \neg m)$$

Backtrack level determined by analyzing conflict clause

Conflict clause: $j \vee \neg d \vee k$

Maximum level different from conflict level

$\Rightarrow 3$



Restarts (after conflicts)

Concept

- Frequently restart (from the beginning) with new (learned) information (Gomes et al., 1997)
- After conflict clause determined:
  - Forget current truth assignment
  - But: keep learned clauses

Idea

- Sometimes “stuck deep in the search”(quite some variables assigned)

Restarts (after conflicts)

Concept

- Frequently restart (from the beginning) with new (learned) information (Gomes et al., 1997)
- After conflict clause determined:
  - Forget current truth assignment
  - But: keep learned clauses

Idea

- Sometimes “stuck deep in the search”(quite some variables assigned)

Restarts (after conflicts)

Algorithm 4: CDCL(F, τ)

```
while True do
    Choose x ∈ var(F_τ), a ∈ {0,1}                          /* Heuristic */
    if Nothing to Decide then
        return SAT
    F := Simplify(F_τ)                                       /* UP */
    while F contains dissatisfied clause C_dissat do
        C_confl := CompConflictCls(C_dissat)                 /* 1st UIP */
        if C_confl = ⊥ then
            return UNSAT
        if m conflicts occurred then
            Backtrack to level 0                             /* Restarts */
        else
            Backtrack(C_conflict)
        F := Simplify(F_τ)                                   /* UP */
```

Forgetting

Situation

- New clauses learned
- Memory wasted with unnecessary clauses
- Cannot store arbitrarily many learned clauses

Idea

- Periodically delete very large learned clauses
  - Not used recently or frequently in deriving new clauses
- Versions:
  - Periodically random selection (larger than threshold)
  - Backtracking leaves many literals unassigned (likely never get used)
  - LBD: num. of different decision levels in clause (prefer small LBD)

Forgetting

Situation

- New clauses learned
- Memory wasted with unnecessary clauses
- Cannot store arbitrarily many learned clauses

Idea

- Periodically delete very large learned clauses
  - Not used recently or frequently in deriving new clauses
- Versions:
  - Periodically random selection (larger than threshold)
  - Backtracking leaves many literals unassigned (likely never get used)
  - LBD: num. of different decision levels in clause (prefer small LBD)

Simplification Rules

Unit Propagation (UP) (Dowling and Gallier, 1984)

$$\frac{\ell \vee \ell_1 \vee \ldots \vee \ell_m}{\ell}$$

Efficient Implementation

- Best:
  - Clause checked *only* after all but one literal assigned 0
  - No check if clause satisfied or more than one literal unassigned
- Two watched literals (Moskewicz et al., 2001)
  - Clause represented by two unassigned literals
  - Check clause only if one of them assigned 0

Dynamic Branching Heuristic

Algorithm 5: CDCL(F, τ)

```
while True do
    Choose x ∈ var(F_τ), a ∈ {0,1}                    /* Heuristic */
    if Nothing to Decide then
        return SAT
    F := Simplify(F_τ)                                  /* UP */
    while F contains dissatisfied clause C_dissat do
        C_confl := CompConflictCls(C_dissat)            /* 1st UIP */
        if C_confl = ⊥ then
            return UNSAT
        if m steps carried out then
            Backtrack to level 0                        /* Restarts */
        else
            Backtrack(C_conflict)
        F := Simplify(F_τ)                              /* UP */
```

Dynamic Branching Heuristic

Idea / Effects

- Variables:
  - minimize search space
- Values: (Pipatsrisawat and Darwiche, 2010)
  - guide search towards a solution (or conflict)
  - cache solutions of subproblems (Phase Saving/RSAT)

Variable State Independent Decaying Sum (VSIDS) (Moskewicz et al., 2001)

- Introduced with zChaff
- Idea: Literals that cause many conflicts assigned first
- Assign a score to each literal
- Decision: Take literal with highest score
- Conflict: Increase score of each conflict literal
- Frequently: after k decisions divide each score by ℓ

Dynamic Branching Heuristic

Idea / Effects

- Variables:
  - minimize search space
- Values: (Pipatsrisawat and Darwiche, 2010)
  - guide search towards a solution (or conflict)
  - cache solutions of subproblems (Phase Saving/RSAT)

Variable State Independent Decaying Sum (VSIDS) (Moskewicz et al., 2001)

- Introduced with zChaff
- Idea: Literals that cause many conflicts assigned first
- Assign a score to each literal
- Decision: Take literal with highest score
- Conflict: Increase score of each conflict literal
- Frequently: after k decisions divide each score by ℓ

Pre-/Inprocessing

Situation

- Large formulas
- Various variables/clauses are totally irrelevant for decision on satisfiability of formula

Idea

- Automated simplification (rewriting) of formula
- Eliminate unnecessary variables/clauses

Requirements

- Satisfiability preserved
- Rewriting works fast (linear-time); slow (quadratic) does usually not pay off

Types

- Preprocessing (runs in the beginning)
- Inprocessing (takes learned clauses into account)

(Järvisalo et al., 2012)

Pre-/Inprocessing

Situation

- Large formulas
- Various variables/clauses are totally irrelevant for decision on satisfiability of formula

Idea

- Automated simplification (rewriting) of formula
- Eliminate unnecessary variables/clauses

Requirements

- Satisfiability preserved
- Rewriting works fast (linear-time); slow (quadratic) does usually not pay off

Types

- Preprocessing (runs in the beginning)
- Inprocessing (takes learned clauses into account)

(Järvisalo et al., 2012)

Pre-/Inprocessing

Situation

- Large formulas
- Various variables/clauses are totally irrelevant for decision on satisfiability of formula

Idea

- Automated simplification (rewriting) of formula
- Eliminate unnecessary variables/clauses

Requirements

- Satisfiability preserved
- Rewriting works fast (linear-time); slow (quadratic) does usually not pay off

Types

- Preprocessing (runs in the beginning)
- Inprocessing (takes learned clauses into account) (Järvisalo et al., 2012)

Pre-/Inprocessing

Situation

- Large formulas
- Various variables/clauses are totally irrelevant for decision on satisfiability of formula

Idea

- Automated simplification (rewriting) of formula
- Eliminate unnecessary variables/clauses

Requirements

- Satisfiability preserved
- Rewriting works fast (linear-time); slow (quadratic) does usually not pay off

Types

- Preprocessing (runs in the beginning)
- Inprocessing (takes learned clauses into account) (Järvisalo et al., 2012)

Pre-/Inprocessing

Preprocessing

- Can be extremely beneficial
- Polynomial, but can often not be run until completion
- Runs only once

Inprocessing

- Allows costly preprocessors
- Resume preprocessing between restarts
- Limit preprocessing time in relation to search time
- Equivalences learned in search can be exploited
- Interesting for encoding optimizations
- “Recovers” some structural properties of the original formula/problem (lost when encoding in CNF)

Preprocessing (Bounded Variable Elimination)

Bounded Variable Elimination

- Based on resolution
- Implemented in most solvers
- Complexity (eliminating one variable): quadratic
- Apply only if increment in size is small
- Restricted to steps which *do not increase* the number of clauses (SatElite)

(Eén and Biere, 2005)

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

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

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

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

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

$$\overbrace{(\neg a \vee b) \wedge (\neg a \vee c) \wedge (a \vee \neg b \vee \neg c)}^{a \leftrightarrow b \wedge c} \wedge (a \vee d) \wedge (a \vee e) \wedge (\neg a \vee f) \wedge (\neg a \vee g)$$

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

$$(a \vee d) \wedge (a \vee e) \wedge (\neg a \vee f) \wedge (\neg a \vee g)$$

$$a \leftrightarrow b \wedge c$$

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

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

$$(a \vee d) \wedge (a \vee e) \wedge (\neg a \vee f) \wedge (\neg a \vee g)$$

$$a \leftrightarrow b \wedge c$$

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

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

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

$$(a \vee d) \wedge (a \vee e) \wedge (\neg a \vee f) \wedge (\neg a \vee g)$$

$$a \leftrightarrow b \wedge c$$

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

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

Preprocessing (Bounded Variable Elimination)

Method

- Eliminate some variable a
  - Take all clauses of F that contain a and $\neg a$, resp.
  - Extract functional dependencies
  - Substitute with functional dependency

Example

$(a \vee d) \wedge (a \vee e) \wedge (\neg a \vee f) \wedge (\neg a \vee g)$

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

In contrast:
Simply resolving yields additionally

$(c \vee e) \wedge (c \vee f) \wedge (d \vee e) \wedge (d \vee f)$

Preprocessing (Fast Subsumption)

Method

- Subsumed clause is redundant
- Eliminate subsumed clauses

$$\frac{\ell_1 \vee \ldots \vee \ell_k \quad \ell_1 \vee \ldots \vee \ell_m}{\ell_1 \vee \ldots \vee \ell_k} \qquad k \leq m$$

- Idea: removes "trash" produced by bounded variable elimination

Preprocessing (Blocked Clauses)

Blocked Clauses (Kullmann, 1999)

- Blocked literal ℓ: Clause $\ell \vee \ell_1 \vee \ldots \vee \ell_m$ *blocked by* ℓ if for every clause $\neg\ell \vee \ell'_1 \vee \ldots \vee \ell'_n$ resolving

$$\frac{\ell_1 \vee \ldots \vee \ell_m \vee \ell, \quad \ell'_1 \vee \ldots \vee \ell'_n \vee \neg\ell}{\ell_1 \vee \ldots \vee \ell_m \vee \ell'_1 \vee \ldots \vee \ell'_n}$$

  yields tautology
- Blocked clause: if it contains *a literal* that blocks it
- Removing does not effect satisfiability

Example

$(a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$

Preprocessing (Blocked Clauses)

Blocked Clauses (Kullmann, 1999)

- Blocked literal ℓ: Clause $\ell \vee \ell_1 \vee \ldots \vee \ell_m$ *blocked by* ℓ if for every clause $\neg\ell \vee \ell'_1 \vee \ldots \vee \ell'_n$ resolving

$$\frac{\ell_1 \vee \ldots \vee \ell_m \vee \ell, \quad \ell'_1 \vee \ldots \vee \ell'_n \vee \neg\ell}{\ell_1 \vee \ldots \vee \ell_m \vee \ell'_1 \vee \ldots \vee \ell'_n}$$

  yields tautology
- Blocked clause: if it contains *a literal* that blocks it
- Removing does not effect satisfiability

Example

$(a \vee b) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$

Preprocessing (Blocked Clauses)

Blocked Clauses (Kullmann, 1999)

- Blocked literal ℓ: Clause $\ell \vee \ell_1 \vee \ldots \vee \ell_m$ *blocked by* ℓ if for every clause $\neg\ell \vee \ell'_1 \vee \ldots \vee \ell'_n$ resolving

$$\frac{\ell_1 \vee \ldots \vee \ell_m \vee \ell, \quad \ell'_1 \vee \ldots \vee \ell'_n \vee \neg\ell}{\ell_1 \vee \ldots \vee \ell_m \vee \ell'_1 \vee \ldots \vee \ell'_n}$$

  yields tautology
- Blocked clause: if it contains *a literal* that blocks it
- Removing does not effect satisfiability

Example

$$\overbrace{(a \vee b)}^{\text{not blocked}} \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee c)$$

Preprocessing (Blocked Clauses)

Blocked Clauses (Kullmann, 1999)

- Blocked literal ℓ: Clause $\ell \vee \ell_1 \vee \ldots \vee \ell_m$ *blocked by* ℓ if for every clause $\neg\ell \vee \ell'_1 \vee \ldots \vee \ell'_n$ resolving

$$\frac{\ell_1 \vee \ldots \vee \ell_m \vee \ell, \quad \ell'_1 \vee \ldots \vee \ell'_n \vee \neg\ell}{\ell_1 \vee \ldots \vee \ell_m \vee \ell'_1 \vee \ldots \vee \ell'_n}$$

  yields tautology
- Blocked clause: if it contains *a literal* that blocks it
- Removing does not effect satisfiability

Example

$$\overbrace{(a \vee b)}^{\text{not blocked}} \wedge \overbrace{(a \vee \neg b \vee \neg c)}^{\text{blocked by } a \text{ and } \neg c} \wedge (\neg a \vee c)$$

Preprocessing (Blocked Clauses)

Blocked Clauses (Kullmann, 1999)

- Blocked literal ℓ: Clause $\ell \vee \ell_1 \vee \ldots \vee \ell_m$ *blocked by* ℓ if for every clause $\neg\ell \vee \ell'_1 \vee \ldots \vee \ell'_n$ resolving

$$\frac{\ell_1 \vee \ldots \vee \ell_m \vee \ell, \quad \ell'_1 \vee \ldots \vee \ell'_n \vee \neg\ell}{\ell_1 \vee \ldots \vee \ell_m \vee \ell'_1 \vee \ldots \vee \ell'_n}$$

  yields tautology
- Blocked clause: if it contains *a literal* that blocks it
- Removing does not effect satisfiability

Example

$$\overbrace{(a \vee b)}^{\text{not blocked}} \wedge \overbrace{(a \vee \neg b \vee \neg c)}^{\text{blocked by } a \text{ and } \neg c} \wedge \overbrace{(\neg a \vee c)}^{\text{blocked by } c}$$

Techniques used for Pre-/Inprocessing

- Variables
  - Unit Propagation (UP); apply whenever possible
  - Pure Literals
  - Equivalent literals
  - Failed literals and generalizations
  - Tautologies
  - Subsumed clauses
  - Self-subsuming resolution
  - Bounded variable elimination
- Clauses
  - Blocked clauses
- Binary clauses: Transitive Reductions
- Binary clauses: Hidden tautology elimination
- Adding Redundancy
  - Hyper binary resolution
  - Bounded variable addition
  - Clause addition

Summary & Learning Goals

FIXME

References I

Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *J. Artif. Intell. Res.*, 40:353–373, 2011. doi: 10.1613/JAIR.3152. URL https://doi.org/10.1613/jair.3152.

Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *J. of the ACM*, 7(3):201–215, 1960. doi: 10.1145/321033.321034.

Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, July 1962. ISSN 0001-0782. doi: 10.1145/368273.368557.

William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *J. Logic Programming*, 1(3):267–284, 1984. ISSN 0743-1066. doi: 10.1016/0743-1066(84)90014-1.

References II

Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer Verlag, 2005.

Johannes K. Fichte, Daniel Le Berre, Markus Hecher, and Stefan Szeider. The silent (r)evolution of sat. *Commun. ACM*, 66(6):64–72, May 2023. ISSN 0001-0782. doi: 10.1145/3560469.

CarlaP. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In Gert Smolka, editor, *Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming (CP'97)*, volume 1330 of *Lecture Notes in Computer Science*, pages 121–135, Linz, Austria, 1997. Springer Verlag. ISBN 978-3-540-63753-0. doi: 10.1007/BFb0017434. URL http://dx.doi.org/10.1007/BFb0017434.

References III

Matti Järvisalo, MarijnJ.H. Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer Verlag, 2012. ISBN 978-3-642-31364-6. doi: 10.1007/978-3-642-31365-3_28. URL http://dx.doi.org/10.1007/978-3-642-31365-3_28.

Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In Karem A. Sakallah and Laurent Simon, editors, *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT'11)*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer Verlag, Ann Arbor, MI, USA, June 2011. ISBN 978-3-642-21580-3. doi: 10.1007/978-3-642-21581-0_27.

Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of cnf's based on short tree-like resolution proofs. *Electronic Colloquium on Computational Complexity (ECCC)*, 6(41), 1999.

References IV

Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient sat solver. In Jan Rabaey, editor, *Proceedings of the 38th Annual Design Automation Conference (DAC'01)*, pages 530–535, New York, NY, USA, 2001. Assoc. Comput. Mach., New York. ISBN 1-58113-297-2. doi: 10.1145/378239.379017. URL http://doi.acm.org/10.1145/378239.379017.

Jakob Nordström. Narrow proofs may be spacious: Separating space and width in resolution. In Jon Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 507–516, Seattle, WA, USA, May 2006. Assoc. Comput. Mach., New York. ISBN 1-59593-134-1. doi: 10.1145/1132516.1132590.

K. Pipatsrisawat and A. Darwiche. *Rsat*, 2010. URL http://reasoning.cs.ucla.edu/rsat/.

References V

João P. Marques Silva and Karem A. Sakallah. GRASP - a new search algorithm for satisfiability. In *Proceedings on the 7th International Conference on Computer-Aided Design (ICCAD'96)*, pages 220–227, San Jose, CA, USA, November 1996. Assoc. Comput. Mach., New York.

Niklas Sörensson and Armin Biere. Minimizing learned clauses. In Oliver Kullmann, editor, *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT'09)*, volume 5584 of *Lecture Notes in Computer Science*, pages 237–243. Springer Verlag, Swansea, UK, June / July 2009. ISBN 978-3-642-02776-5. doi: 10.1007/978-3-642-02777-2_23. URL http://dx.doi.org/10.1007/978-3-642-02777-2_23.

Backup Slides

Conflict-Clause Minimization (Sörensson and Biere, 2009)

Self-Subsumption

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee \ldots \vee \ell_m \vee \mathbf{x}, \quad \ell_1 \vee \ldots \vee \ell_k \vee \neg\mathbf{x}}{\ell_1 \vee \ldots \vee \ell_m} \qquad k \leq m$$

- Keep $\ell_1 \vee \ldots \vee \ell_m$ instead of $\ell_1 \vee \ldots \vee \ell_k \vee \ell_m \vee \mathbf{x}$
- Resolve conflict clause with known implications
- Shorten conflict clause

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$

$$(\neg a \vee \neg c \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$

$$(\neg a \vee \neg c \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$

$$(\neg a \vee \neg c \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$

$$(\downarrow \neg a \vee \downarrow \neg c \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$
Assigned Literals: None

$$(\overset{\downarrow}{\neg a} \vee \overset{\downarrow}{\neg c} \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c$

Assigned: c

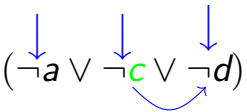
$$(\underset{\downarrow}{\neg a} \vee \underset{\downarrow}{\neg c} \vee \neg d)$$

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg c \rightarrow \neg d$

Assigned: c

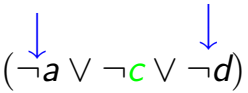


Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg d$

Assigned: c



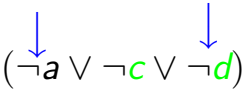
$\neg c$ no longer checked

Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg d$

Assigned: c, d

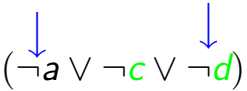


Simplification Rules (Watched Lits)

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

Watched Literals: $\neg a$ and $\neg d$

Assigned: c



propagate a
(to satisfy the clause)

Simplified Model of SAT solvers

In detail

- $\texttt{CDCL}^{\text{simp}}$ restarts *very* often
- completely random branching (often enough)

Results (Atserias et al., 2011)

- Assume small resolution width.
- CDCL solver will provably be fast

Implications

- Solve formulas of small treewidth fast

Limits

- Small width does not imply small space (Nordström, 2006)
- Memory consumption very important bottleneck in practice