SATisfiability Solving: How do SAT solvers work?

Ruben Martins

University of Oxford

February 6, 2014

Thanks

Joao Marques-Silva:

- Providing some of the slides given in this talk
- SAT Summer School 2013, "CDCL SAT Solvers & SAT-Based Problem Solving":
  `http://satsmt2013.ics.aalto.fi/slides/Marques-Silva.pdf`

The Success of SAT

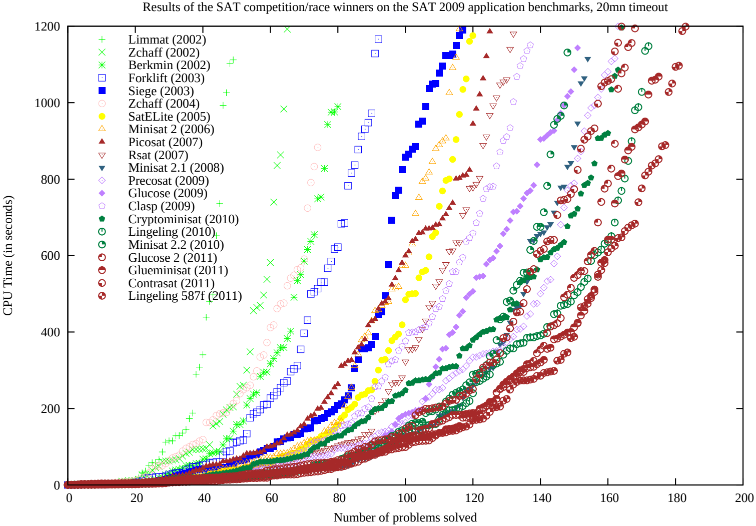
- Well-known NP-complete decision problem [C71]

The Success of SAT

- Well-known NP-complete decision problem [C71]
- In practice, SAT is a success story of Computer Science
  - Hundreds (even more?) of practical applications

SAT Solver Improvement

[Source: Le Berre&Biere 2011]



Outline

- Basic Definitions
- DPLL
- CDCL
  - Features
  - Performance
  - Why do CDCL solvers work in practice?

Preliminaries

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

Preliminaries

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

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

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

Resolution

- Resolution rule: [DP60,R65]

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

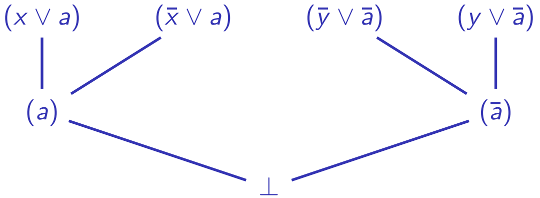
  - Complete proof system for propositional logic

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



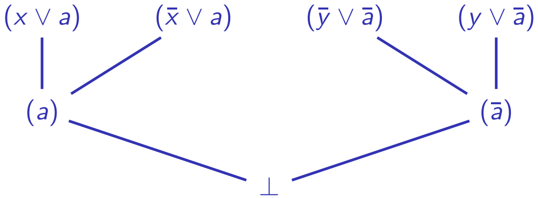
  - Extensively used with (CDCL) SAT solvers

Resolution

- Resolution rule: [DP60,R65]

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

  - Complete proof system for propositional logic



  - Extensively used with (CDCL) SAT solvers

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

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

  - (α) subsumes $(\alpha \vee x)$

Unit Propagation

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

Unit Propagation

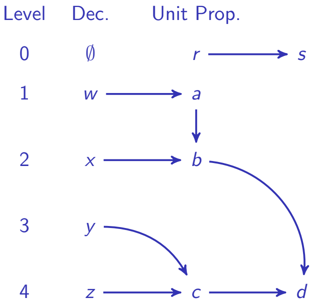
$$\begin{aligned}\mathcal{F} \;&=\; (r) \wedge (\bar{r} \vee s) \wedge \\ &\quad\; (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\ &\quad\; (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)\end{aligned}$$

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

Unit Propagation

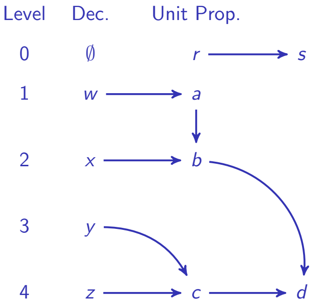
$$
\begin{aligned}
\mathcal{F} \quad = \quad & (r) \wedge (\bar{r} \vee s) \wedge \\
& (\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b) \\
& (\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)
\end{aligned}
$$

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



Unit Propagation

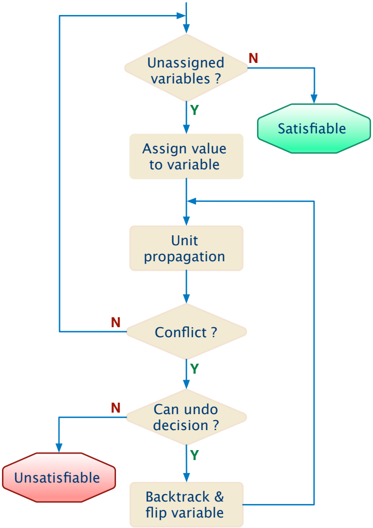
$$\mathcal{F} = (r) \wedge (\bar{r} \vee s) \wedge$$
$$(\bar{w} \vee a) \wedge (\bar{x} \vee \bar{a} \vee b)$$
$$(\bar{y} \vee \bar{z} \vee c) \wedge (\bar{b} \vee \bar{c} \vee d)$$



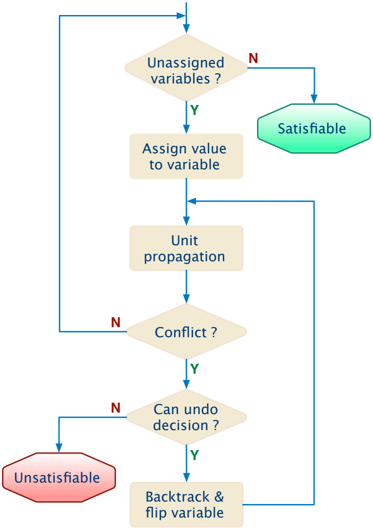
- Decisions / Variable Branchings:
  $w = 1, x = 1, y = 1, z = 1$

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

The DPLL Algorithm

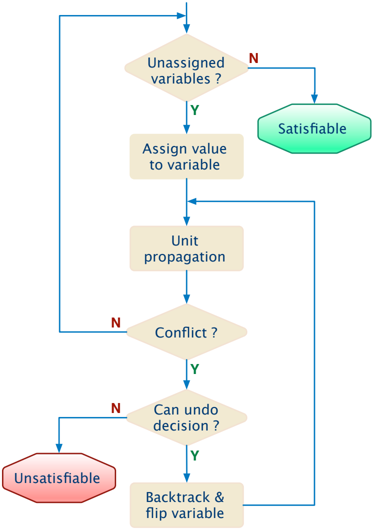


The DPLL Algorithm



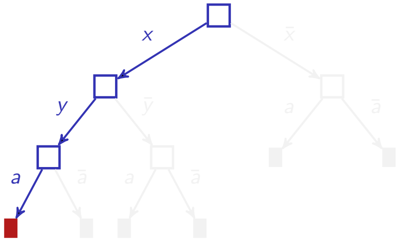
$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

The DPLL Algorithm

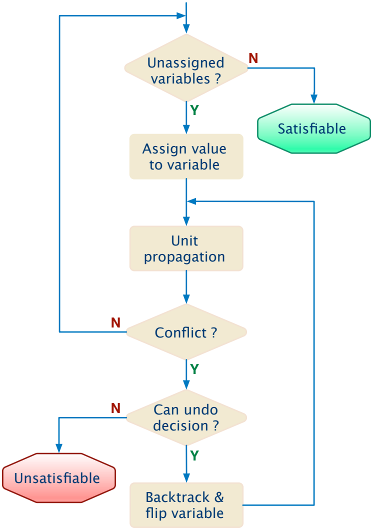


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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

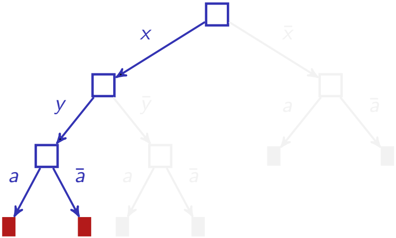


The DPLL Algorithm

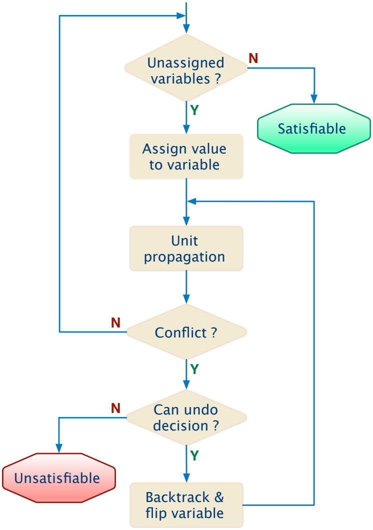


$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$

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

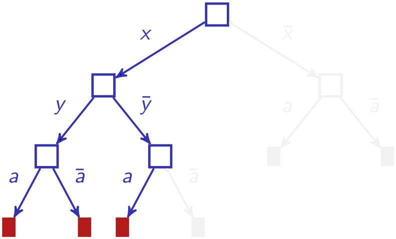


The DPLL Algorithm

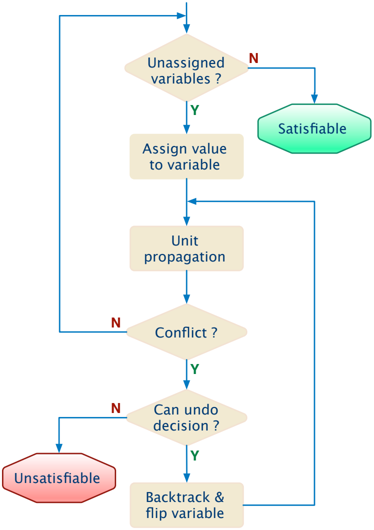


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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

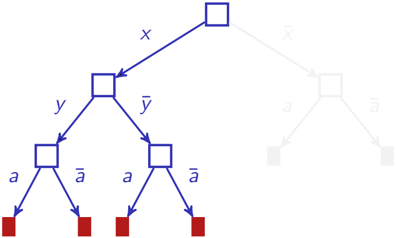


The DPLL Algorithm

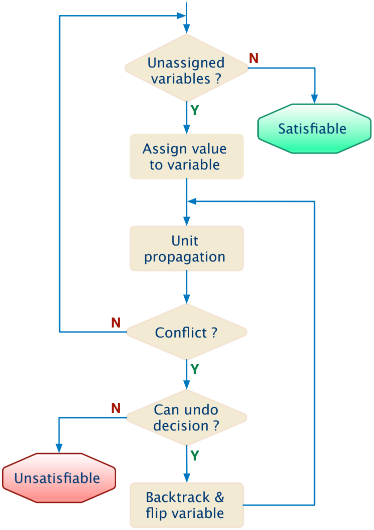


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

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

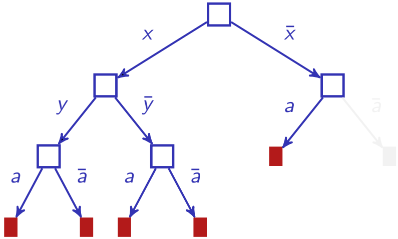


The DPLL Algorithm

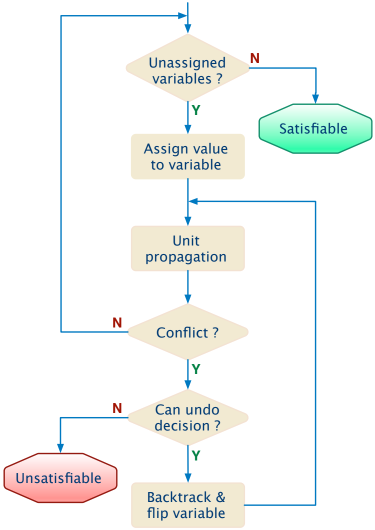


$$\mathcal{F} = (x \vee y) \wedge (a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	$\longrightarrow y$
2	a	$\longrightarrow b \longrightarrow \perp$

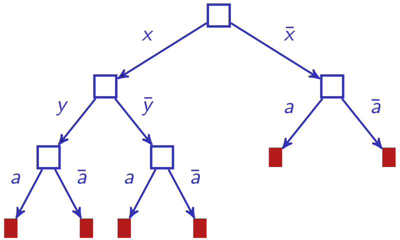


The DPLL Algorithm



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

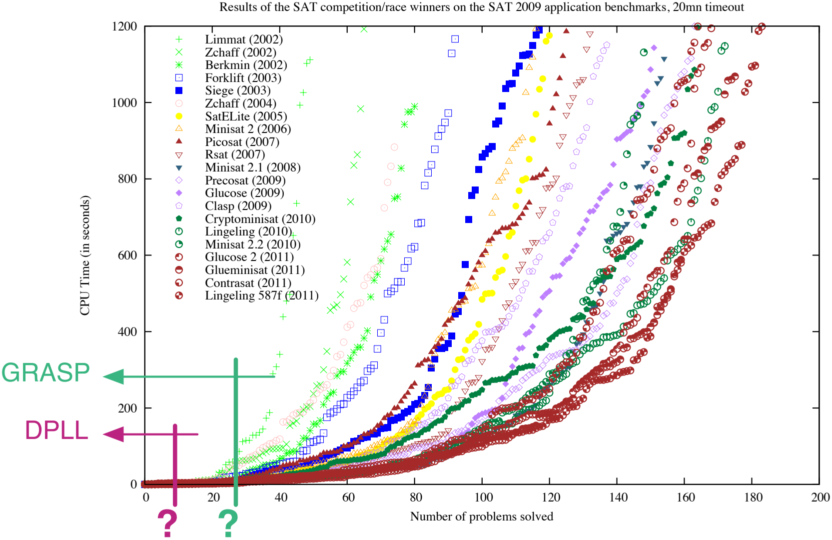
Level	Dec.	Unit Prop.
0	$\emptyset$	
1	$\bar{x}$	$\longrightarrow y$
2	$\bar{a}$	$\longrightarrow \bar{b} \longrightarrow \perp$



What is a CDCL SAT Solver?

- Extend DPLL SAT solver with: [DP60,DLL62]
  - Clause learning & non-chronological backtracking [MSS96,BS97,Z97]
  - Search restarts [GSK98,BMS00,H07,B08]
  - Lazy data structures
    - Watched literals [MMZZM01]
  - Conflict-guided branching
  - ...

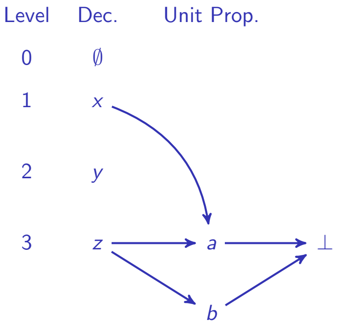
How Significant are CDCL SAT Solvers?



Clause Learning

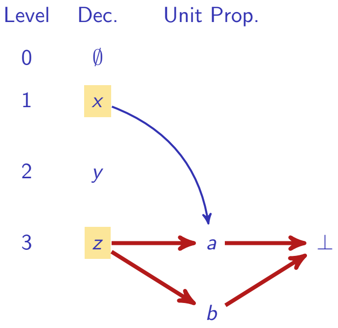
$$\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & \\
0 & \emptyset & & \\
1 & x \arrow[ddr, bend left] & & \\
2 & y & & \\
3 & z \arrow[r] \arrow[dr] & a \arrow[r] & \bot \\
 & & b \arrow[ur] &
\end{tikzcd}$$

Clause Learning



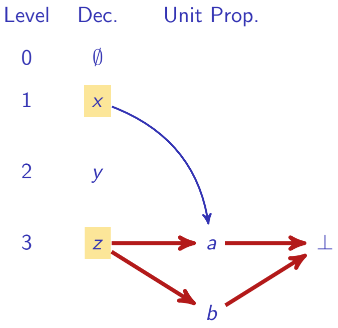
- Analyze conflict

Clause Learning



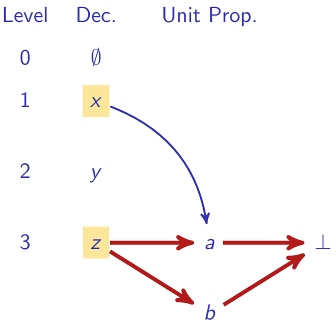
- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels

Clause Learning



- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$

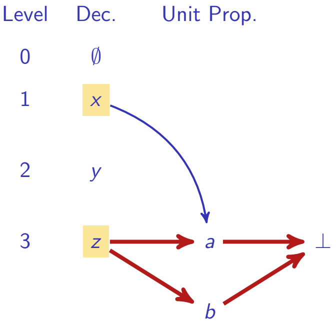
Clause Learning

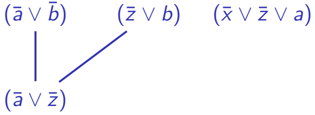


$(\bar{a} \vee \bar{b})$ $(\bar{z} \vee b)$ $(\bar{x} \vee \bar{z} \vee a)$

- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

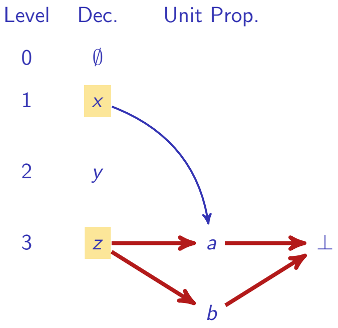
Clause Learning

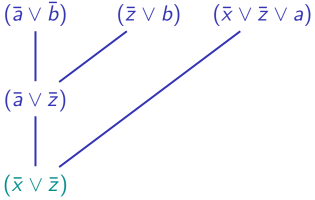




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

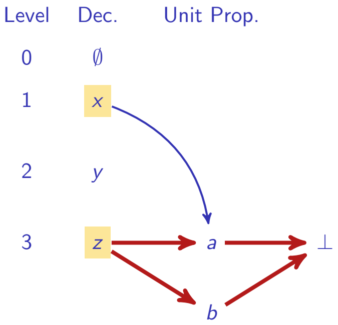
Clause Learning

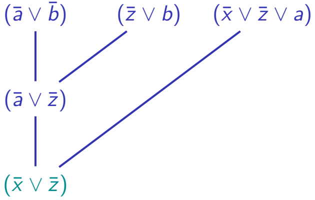




- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution

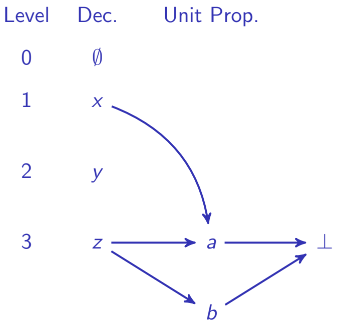
Clause Learning



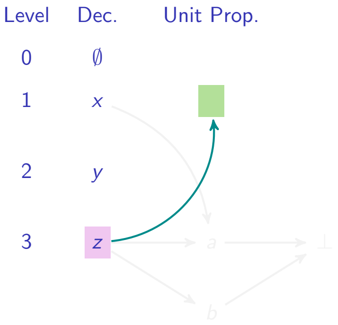


- Analyze conflict
  - Reasons: x and z
    - Decision variable & literals assigned at lower decision levels
  - Create **new** clause: $(\bar{x} \vee \bar{z})$
- Can relate clause learning with resolution
  - Learned clauses result from (**selected**) resolution operations

Clause Learning – After Bracktracking

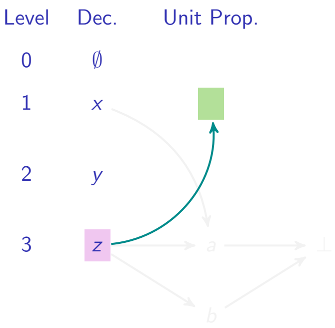


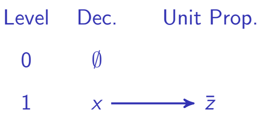
Clause Learning – After Bracktracking



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

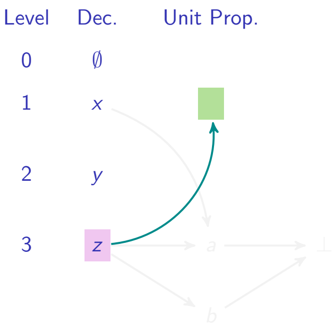
Clause Learning – After Bracktracking

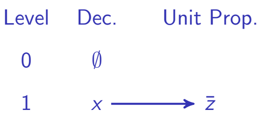




- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1

Clause Learning – After Bracktracking



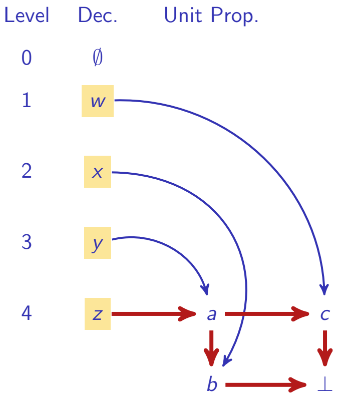


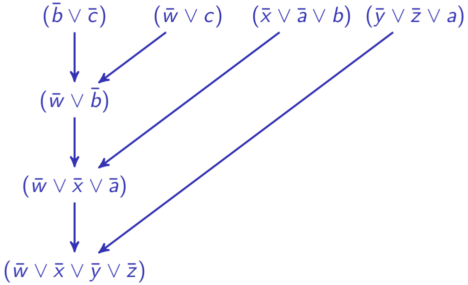
- Clause $(\bar{x} \vee \bar{z})$ is asserting at decision level 1
- Learned clauses are always asserting [MSS96,MSS99]
- Backtracking differs from plain DPLL:
  - Always bactrack after a conflict [MMZZM01]

Unique Implication Points (UIPs)

$$
\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & \\
0 & \emptyset & & \\
1 & w \arrow[dddrr, bend left] & & \\
2 & x \arrow[dddr, bend left] & & \\
3 & y \arrow[dr, bend left] & & \\
4 & z \arrow[r] & a \arrow[r] \arrow[d] & c \arrow[d] \\
 & & b \arrow[r] & \perp
\end{tikzcd}
$$

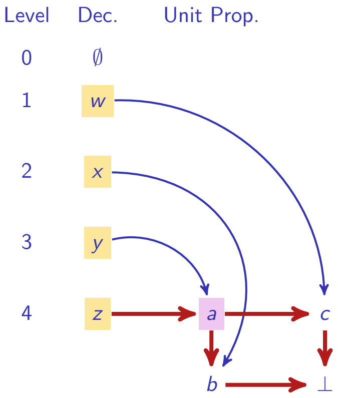
Unique Implication Points (UIPs)

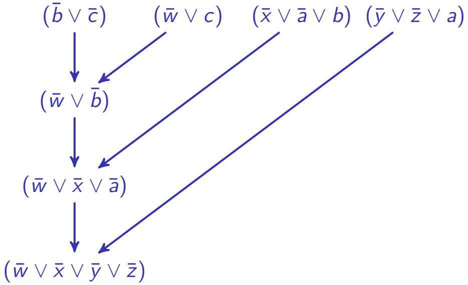




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$

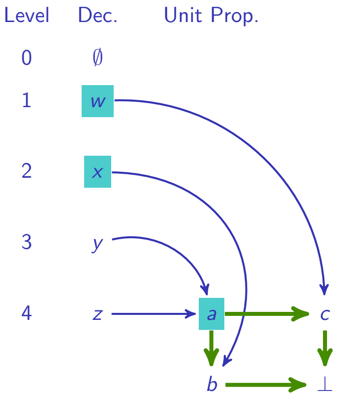
Unique Implication Points (UIPs)

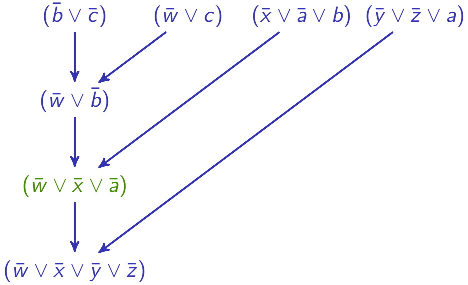




- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$
- But a is an UIP

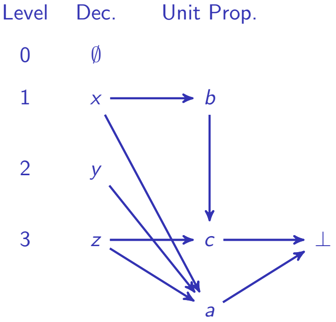
Unique Implication Points (UIPs)



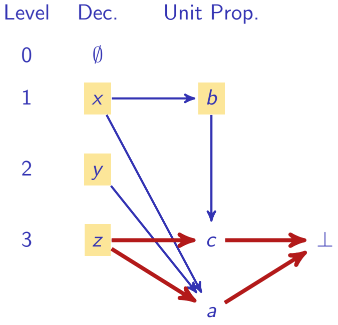


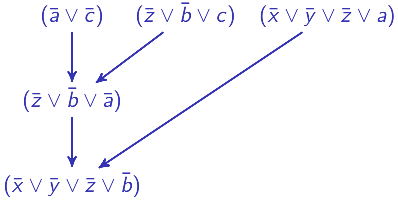
- ~~Learn clause $(\bar{w} \vee \bar{x} \vee \bar{y} \vee \bar{z})$~~
- But a is an UIP
- Learn clause $(\bar{w} \vee \bar{x} \vee \bar{a})$

Clause Minimization



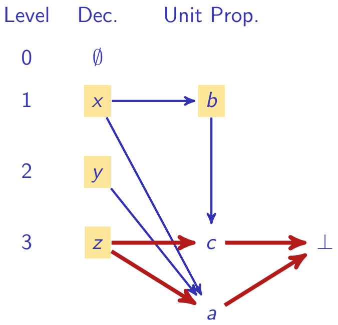
Clause Minimization

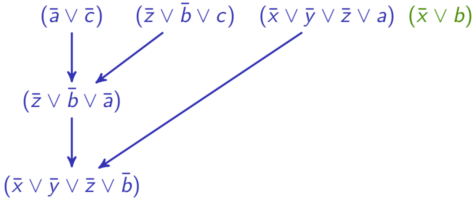




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$

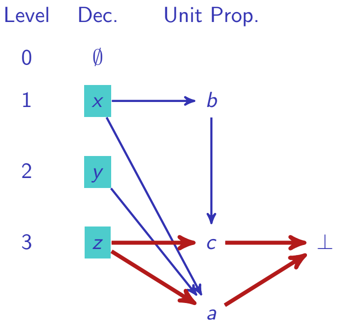
Clause Minimization

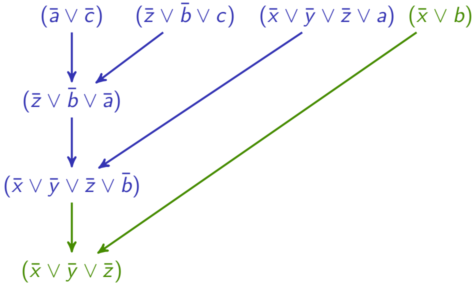




- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$
- Apply self-subsuming resolution (i.e. local minimization) [SB09]

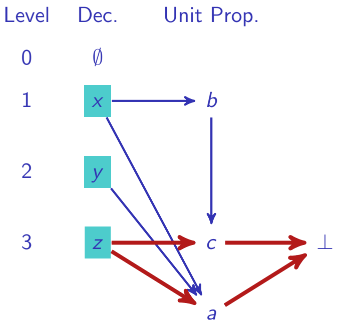
Clause Minimization

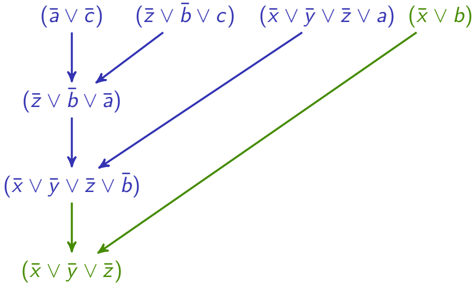




- ~~Learn clause~~ $\cancel{(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})}$
- Apply self-subsuming resolution (i.e. local minimization)

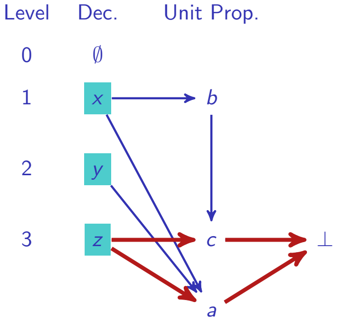
Clause Minimization

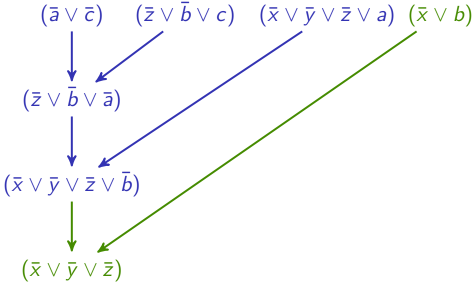




- ~~Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z} \vee \bar{b})$~~
- Apply self-subsuming resolution (i.e. local minimization)
- Learn clause $(\bar{x} \vee \bar{y} \vee \bar{z})$

Clause Minimization



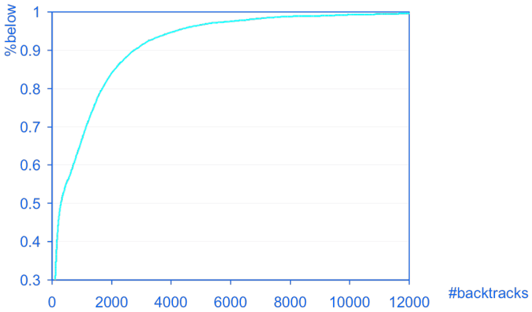


- Other minimization techniques exist:

  e.g Recursive minimization
- Minimization eliminates on average more than 30% of literals

Search Restarts

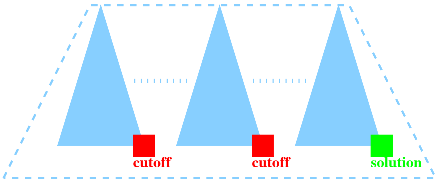
- Heavy-tail behavior: [GSK98]



  - 10,000 runs, branching randomization on industrial instance
    - Use rapid randomized restarts (search restarts)

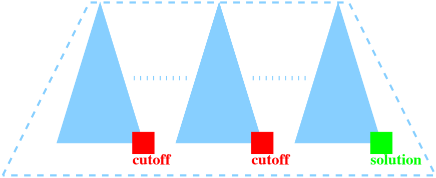
Search Restarts

- Restart search after a number of conflicts



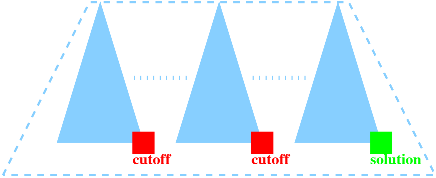
Search Restarts

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



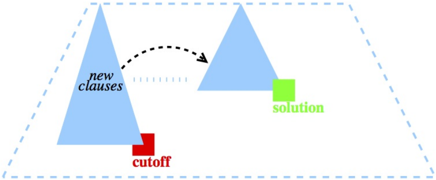
Search Restarts

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



Search Restarts

- Restart search after a number of conflicts
- Increase cutoff after each restart
  - Guarantees completeness
  - Different policies exist
- Works for SAT & UNSAT instances. Why?
- Learned clauses effective after restart(s)



Data Structures Basics

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

Data Structures Basics

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

Data Structures Basics

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

- Clause with k literals results in k references, from literals to the clause

- Number of clause references **equals** number of literals
  - Clause learning can generate **large** clauses

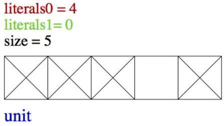
Data Structures Basics

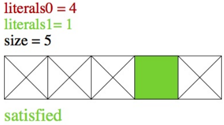
- Each literal l should access clauses containing l
  - Why? Unit propagation
- Clause with k literals results in k references, from literals to the clause
- Number of clause references **equals** number of literals
  - Clause learning can generate **large** clauses
- Clause learning to be effective requires a more efficient representation: **Watched Literals**

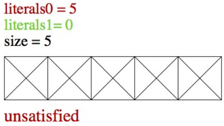
Watched Literals

[MMZZM01]

- Important states of a clause



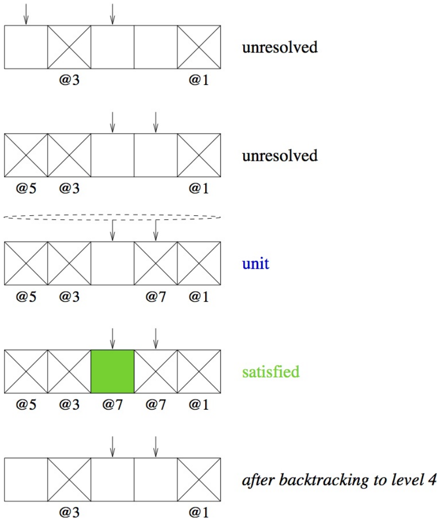




Watched Literals

[MMZZM01]

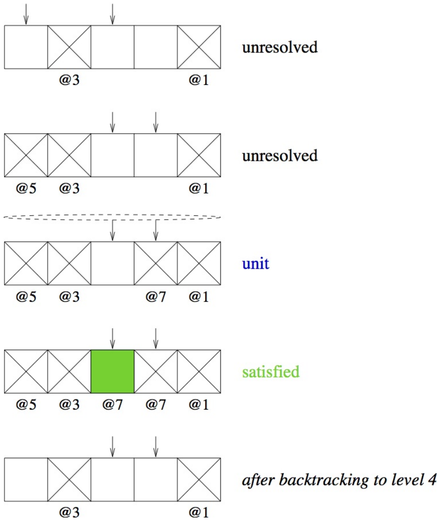
- Important states of a clause
- Associate **2** references with each clause



Watched Literals

[MMZZM01]

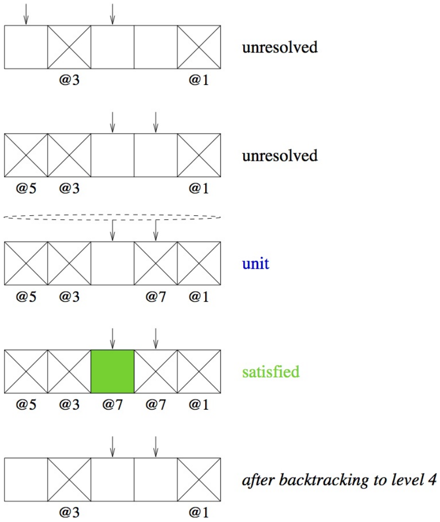
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals



Watched Literals

[MMZZM01]

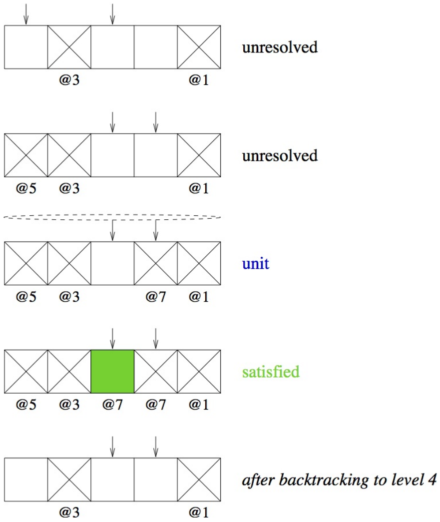
- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals
- References **unchanged** when backtracking



Watched Literals

[MMZZM01]

- Important states of a clause
- Associate **2** references with each clause
- Deciding unit requires traversing all literals
- References **unchanged** when backtracking
- Watched literals are one example of lazy data structures



Additional Key Techniques

- Lightweight branching (VSIDS) [e.g. MMZZM01]
  - Increments the activity of variables that participated in the creation of conflict clauses
  - Pick the literal with the highest activity as the next decision variable

Additional Key Techniques

- Lightweight branching (VSIDS) [e.g. MMZZM01]
  - Increments the activity of variables that participated in the creation of conflict clauses
  - Pick the literal with the highest activity as the next decision variable

- Clause deletion policies
  - Not practical to keep all learned clauses
  - Delete less used clauses [e.g. MSS96,GN02,ES03]

How important is each feature of a CDCL solver?

- CDCL solvers share **four major features**:
  - Conflict-driven clause learning
  - Search Restarts
  - Unit propagation using watched literals
  - Conflict-based branching

- How important is each major feature for the performance of a CDCL solver? [KSMS11]

Empirical study of a CDCL solver

[Source: Katebi, Sakallah & Marques-Silva 2011]

- Experimental approach:
  - MiniSAT as the CDCL solver
  - 1,000 benchmarks from 12 application areas, since early 1990s:
    - Circuit testing (`atpg`), Bioinformatics (`bioinf`), product configuration (`config`), cryptanalysis (`crypto`), equivalence checking (`equiv`), FPGA routing (`fpga`), hardware bounded model checking (`hbmc`), hardware verification (`hverif`), network configuration (`netcfg`), planning (`plan`), software verification (`sverif`), term rewriting (`termrw`)
  - Each benchmark: 10 random reorderings of the CNF
  - Disable one feature at a time and compare with the base case
  - For each configuration count the number of instances solved in under 1,000 seconds

Impact of Conflict-driven clause learning

Family	Runs	$\neg$CL	CDCL
atpg	1,000	965	1,000
bioinf	300	19	150
config	500	472	500
crypto	300	52	237
equiv	300	50	231
fpga	500	325	470
hbmc	2,500	762	2,333
hverif	2,000	1,413	1,984
netcfg	100	0	87
plan	800	327	650
sverif	1,200	336	1,006
termrw	500	116	420
Total	10,000	4,827	9,068

Impact of Conflict-based branching

- Variable State Independent Decaying Sum (VSIDS)
- Dynamic Largest Individual Sum (DLIS):
  - Each literal has a counter with the number of times it appears in unresolved clauses
  - Pick the literal with the highest sum as the next decision literal
- Random Heuristic

Impact of Conflict-based branching

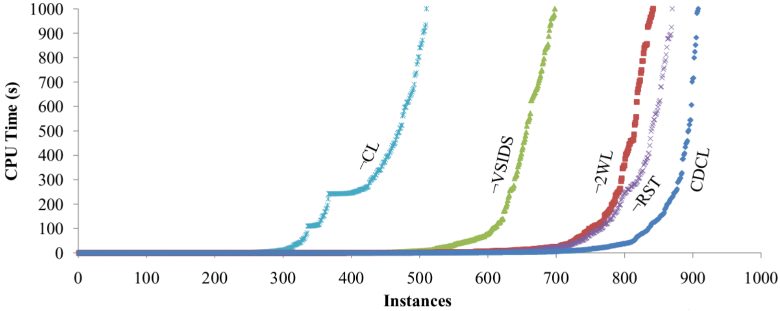
Family	Runs	DLIS	RDM	CDCL (VSIDS)
`atpg`	1,000	1,000	1,000	1,000
`bioinf`	300	34	46	150
`config`	500	500	500	500
`crypto`	300	22	35	237
`equiv`	300	92	162	231
`fpga`	500	403	421	470
`hbmc`	2,500	1,872	2,057	2,333
`hverif`	2,000	1,700	1,949	1,984
`netcfg`	100	20	72	87
`plan`	800	449	490	650
`sverif`	1,200	592	302	1,006
`termrw`	500	248	291	420
Total	10,000	6,932	7,325	9,068

Impact of Watched Literals and Search Restarts

Family	Runs	¬2WL	¬RST	CDCL
`atpg`	1,000	1,000	1,000	1,000
`bioinf`	300	88	141	150
`config`	500	500	500	500
`crypto`	300	113	235	237
`equiv`	300	187	224	231
`fpga`	500	444	441	470
`hbmc`	2,500	2,241	2,307	2,333
`hverif`	2,000	1,934	1,967	1,984
`netcfg`	100	60	74	87
`plan`	800	559	564	650
`sverif`	1,200	937	754	1,006
`termrw`	500	346	446	420
Total	10,000	8,409	8,653	9,068

Empirical study of a CDCL solver

[Source: Katebi, Sakallah & Marques-Silva 2011]



- Importance of major features:
  - CL > VSIDS > 2WL > RST
  - Combination of all four features yields best performance

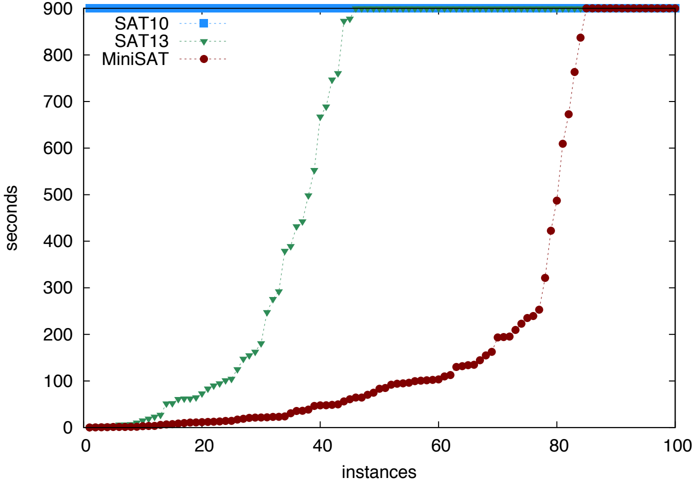
How easy is to make a SAT solver?

Donald Knuth:

- Professor Emeritus at Stanford University
- Author of several books, $\TeX$, ...
- Wrote several small SAT solvers for Volume 4B of the new edition of "The Art of Computer Programming":
  - `http://www-cs-faculty.stanford.edu/~knuth/programs.html`
  - DPLL solver (**SAT10**), CDCL solver (**SAT13**)

How easy is to make a SAT solver?

100 benchmarks, SAT Race 2008, 900 seconds time limit



Which SAT solver should I use?

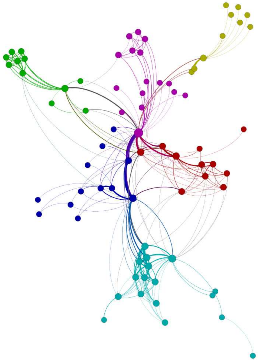
- Using as a black-box:
  - Check the SAT competition: `http://www.satcompetition.org/`
  - `Lingeling`, `Glucose` perform well
- Using the API or changing the source code:
  - `MiniSAT` (simple and easy to extend)
  - `Glucose` (based on MiniSAT with better performance)
- Disclaimer **!**
  - **Patent** on Chaff: US 20030084411 A1
  - Watched Literals and VSIDS heuristic

Why do CDCL solvers work in practice?

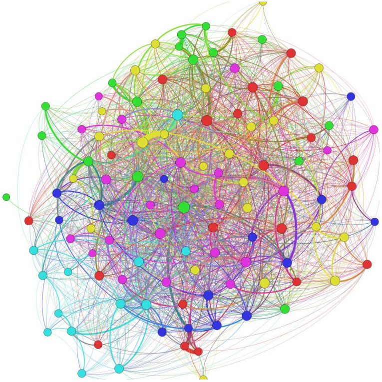
- CL as powerful as general resolution (RES) [PD09]
- In practice:
  - RES impractical in practice
  - CL very effective in practice
- So, why does CL work in practice?
  - Clause learning explained by sequence of (trivial) resolution operations
  - Clause learning (somehow) identifies the right resolution operations
    - From the analysis of conflicts resulting from unit propagation
  - Hard problems can be solved by exploiting structure **!**

Why do CDCL solvers work in practice?

[Source: Ansótegui, Giráldez-Cru & Levy 2012]



Industrial benchmark



Random benchmark

Why do CDCL solvers work in practice?

[W99,ABL09,AGCL12]

- Transform a CNF formula into a graph and analyze its structure
  - For example using the modularity measure [AGCL12]
- Industrial benchmarks have a clear community structure
- Modularity slowly decreases with learned clauses
  - Does not completely destroy the structure of the formula
  - May change the partitions
- Random benchmarks do not have community structure

Conclusions

The **secret ingredients** for having an efficient SAT solver:

Conclusions

The **secret ingredients** for having an efficient SAT solver:

- Make mistakes **!**
  - Learn from your conflicts
  - Perform non-chronological backtracking
  - Restart the search

Conclusions

The **secret ingredients** for having an efficient SAT solver:

- Make mistakes **!**
  - Learn from your conflicts
  - Perform non-chronological backtracking
  - Restart the search
- Be lazy **!**
  - Lazy data structures
  - Lightweight heuristics

Next Talk

- SATisfiability Solving: How to solve problems with SAT?
  - Encoding to CNF
  - Impact of different encodings
  - Successful encoding techniques

Next Talk

- SATisfiability Solving: How to solve problems with SAT?
  - Encoding to CNF
  - Impact of different encodings
  - Successful encoding techniques

- What is the structure of your encoding **?**

References

C71 S. Cook: The Complexity of Theorem-Proving Procedures. STOC 1971: 151-158

DP60 M. Davis, H. Putnam: A Computing Procedure for Quantification Theory. J. ACM 7(3): 201-215 (1960)

R65 J. Robinson: A Machine-Oriented Logic Based on the Resolution Principle. J. ACM 12(1): 23-41 (1965)

SP04 S. Subbarayan, D. Pradhan: NiVER: Non-increasing Variable Elimination Resolution for Preprocessing SAT Instances. SAT 2004: 276-291

EB05 N. Een, A. Biere: Effective Preprocessing in SAT Through Va011: 201-215

DLL62 M. Davis, G. Logemann, D. Loveland: A machine program for theorem-proving. Commun. ACM 5(7): 394-397 (1962)

MSS96 J. Marques-Silva, K. Sakallah: GRASP - a new search algorithm for satisfiability. ICCAD 1996: 220-227

BS97 R. Bayardo Jr., R. Schrag: Using CSP Look-Back Techniques to Solve Real-World SAT Instances. AAAI/IAAI 1997: 203-208

Z97 H. Zhang: SATO: An Efficient Propositional Prover. CADE 1997: 272-275

References

SSS12 A. Sabharwal, H. Samulowitz, M. Sellmann: Learning Back-Clauses in SAT. SAT 2012: 498-499

SB09 N. Sorensson, A. Biere: Minimizing Learned Clauses. SAT 2009: 237-243

VG09 A. Van Gelder: Improved Conflict-Clause Minimization Leads to Improved Propositional Proof Traces. SAT 2009: 141-146

MSS99 J. Marques-Silva, K. Sakallah: GRASP: A Search Algorithm for Propositional Satisfiability. IEEE Trans. Computers 48(5): 506-521 (1999)

GN02 E. Goldberg, Y. Novikov: BerkMin: A Fast and Robust Sat-Solver. DATE 2002: 142-149

GSK98 C. Gomes, B. Selman, H. Kautz: Boosting Combinatorial Search Through Randomization. AAAI 1998: 431-437

BMS00 L. Baptista, J. Marques-Silva: Using Randomization and Learning to Solve Hard Real-World Instances of Satisfiability. CP 2000: 489-494

H07 J. Huang: The Effect of Restarts on the Efficiency of Clause Learning. IJCAI 2007: 2318-2323

B08 A. Biere: PicoSAT Essentials. JSAT 4(2-4): 75-97 (2008)

References

MMZZM01 M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik: Chaff: Engineering an Efficient SAT Solver. DAC 2001: 530-535

KSMS11 H. Katebi, K. Sakallah, J. Marques Silva: Empirical Study of the Anatomy of Modern Sat Solvers. SAT 2011: 343-356

PD09 K. Pipatsrisawat and A. Darwiche: On the Power of Clause-Learning SAT Solvers with Restarts. CP 2009: 654-668

W99 T. Walsh: Search in a Small World. IJCAI 1999: 1172-1177

ABL09 C. Ansótegui, M. Bonet, J. Levy: On the Structure of Industrial SAT Instances. CP 2009: 127-141

AGCL12 C. Ansótegui, J. Giráldez-Cru, J. Levy: The Community Structure of SAT Formulas. SAT 2012: 410-423