Theory-Insight-Guided Empirical Research for Understanding Applied Combinatorial Solving

Jakob Nordström

University of Copenhagen and Lund University

Dagstuhl Seminar 26292
Analysis of Algorithms Beyond the Worst Case
July 16, 2026

Me and My Research

Jakob Nordström
University of Copenhagen
and Lund University
jn@di.ku.dk
www.jakobnordstrom.se

Research interests:

- Computational complexity theory
- Applied combinatorial optimization
- Certifying algorithms / proof logging

Me and My Research

Jakob Nordström
University of Copenhagen
and Lund University
jn@di.ku.dk
www.jakobnordstrom.se

Research interests:

- *Proving exponential lower bounds for* NP*-hard problems*
- *Designing algorithms to solve* NP*-hard problems in (ideally) linear time*
- *Generating machine-verifiable proofs that such algorithms reason correctly*

The Unreasonable Effectiveness of Combinatorial Solving

- Many combinatorial optimization problems considered in practice are NP-hard
- Widely believed to require exponential time
- However, dramatic improvements since the turn of the millennium has led to very efficient applied algorithms for
  - SAT solving [BHvMW21]
  - Constraint programming [RvBW06]
  - Mixed integer linear programming [AW13, BR07]
  - Satisfiability modulo theories (SMT) solving [BHvMW21]
- Can we use complexity theory to understand what is going on inside such combinatorial solvers?!

Theory and Practice Meeting Halfway?

Proof complexity

- Model reasoning used by algorithm as formal proof system
- Study efficient proofs = sequence of reasoning steps reaching correct answers
- Non-existence of efficient proofs imply (strong) impossibility results, but can only be obtained for highly structured problems
- Existence of efficient proofs does not mean such proofs can be found algorithmically
- And modelling with proof systems can easily oversimplify algorithms

Theory and Practice Meeting Halfway?

Proof complexity

- Model reasoning used by algorithm as formal proof system
- Study efficient proofs = sequence of reasoning steps reaching correct answers
- Non-existence of efficient proofs imply (strong) impossibility results, but can only be obtained for highly structured problems
- Existence of efficient proofs does not mean such proofs can be found algorithmically
- And modelling with proof systems can easily oversimplify algorithms

Theory-insight-guided empirical research

- Run computational experiments with solvers
- Use suitable problems from proof complexity literature (so called crafted formulas) and check if performance matches predictions we cannot prove formally
- For "real-world" problems, analyze proofs generated by solvers

Caveats

TL;DR: *The workshop organizers made me do it. . .*

Caveats

TL;DR: *The workshop organizers made me do it...*

Slightly longer version:

- This is a somewhat old line of research [JMNŽ12, EGG$^+$18, EGNV18, KN20]
- Tons of hard work, but not an obviously rapturous reception...
- No proofs — just circumstantial evidence
- Results are open to interpretation/criticism

Caveats

TL;DR: *The workshop organizers made me do it...*

Slightly longer version:

- This is a somewhat old line of research [JMNŽ12, EGG+18, EGNV18, KN20]
- Tons of hard work, but not an obviously rapturous reception...
- No proofs — just circumstantial evidence
- Results are open to interpretation/criticism
- But I personally still believe this type of research makes sense...
- So maybe it is worth talking about it, to see if this can at least generate some interesting discussions?

Contents of This Talk

For this talk, focus on Boolean satisfiability (SAT) solving

1. Crash course on modern SAT solving
2. Compare actual SAT solver performance on crafted benchmark formulas with properties shown in proof complexity literature [EGG$^+$18]
3. Study properties of actual proofs generated when solvers run on "real-world" benchmark formulas [KN20]

The Boolean Satisfiability (SAT) Problem

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge$$
$$(x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

The Boolean Satisfiability (SAT) Problem

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge$$
$$(x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

- Variables should be set to **true** $(= 1)$ or **false** $(= 0)$
- Literal ℓ: variable x or negated variable $\overline{x}$
- Clause $x \vee \overline{y} \vee z$ specifies x or z should be true or y false
- Conjunctions $\wedge$ require that all constraints should hold simultaneously
- Is there a truth value assignment satisfying all clauses?

State-of-the-art SAT Solvers: Ingredients

Basic SAT solving method: backtracking search / DPLL [DP60, DLL62]

Many more ingredients in modern conflict-driven clause learning (CDCL) SAT solvers (as pioneered in [BS97, MS99, MMZ$^+$01]), e.g.:

- Branching or decision heuristic (choice of pivot variables crucial)
- When backtracking, compute explanation for conflict and add to formula as new clause (clause learning)
- Every once in a while, restart from beginning (but save computed info)
- Preprocessing the formula before the search even starts

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers ≈99% of assignments are unit propagations

Variable Assignment Heuristics

Unit propagation

- Suppose current assignment ρ falsifies all literals in $C = \ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ except one (say ℓ_k) — C is unit under ρ
- Then ℓ_k has to be true, so set it to true
- Known as unit progagation or Boolean constraint progagation
- Always propagate if possible — in modern solvers $\approx$99% of assignments are unit propagations

VSIDS (Variable state independent decaying sum)

- When backtracking, score $+1$ for variables "causing conflict"
- Also multiply all scores with factor $\kappa < 1$ — exponential filter rewarding variables involved in recent conflicts
- When no propagations, decide on unassigned variable with highest score

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose we can compute that decisions $x = 1$, $y = 0$, $z = 1$ responsible for conflict
- Then can add $\overline{x} \vee y \vee \overline{z}$ to avoid these decisions being made again — decision learning scheme (this idea goes all the way back to [SS77])
- Nowadays, more sophisticated learning schemes starting with [MS99, MMZ$^+$01]
- Conflict analysis best described as derivation from clauses unit propagating to conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$p \overset{\mathsf{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

$p \overset{\mathsf{d}}{=} 0$

Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

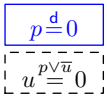
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

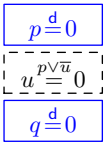
Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

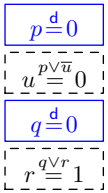
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

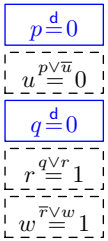
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

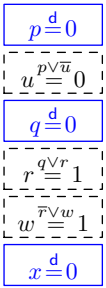
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

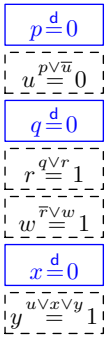
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

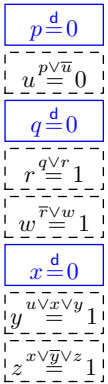
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

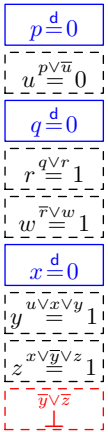
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathsf{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

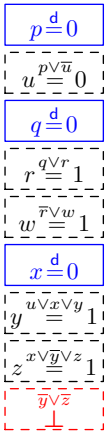
Add to assignment trail

Continue until satisfying assignment or conflict

Decisions, Unit Propagations, and Conflict

Two kinds of assignments — illustrate on example formula:

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Decision

Free choice to assign value to variable

Notation $p \overset{\mathrm{d}}{=} 0$

Unit propagation

Forced choice to avoid falsifying clause

Given $p = 0$, clause $p \vee \overline{u}$ forces $u = 0$

Notation $u \overset{p \vee \overline{u}}{=} 0$ ($p \vee \overline{u}$ is reason clause)

Always propagate if possible, otherwise decide

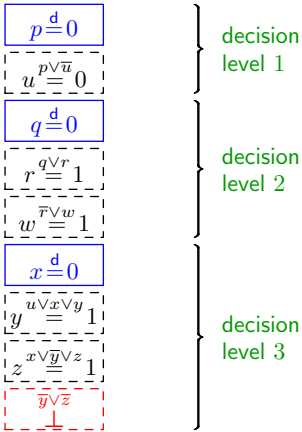
Add to assignment trail

Continue until satisfying assignment or conflict

Conflict Analysis

Time to analyse this conflict and learn from it!

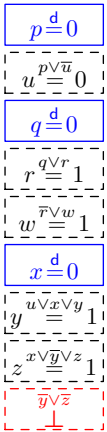
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$

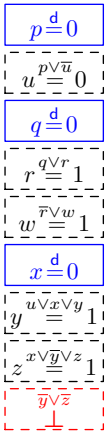


Could backtrack by erasing conflict level & flipping last decision (this is what DPLL does)

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



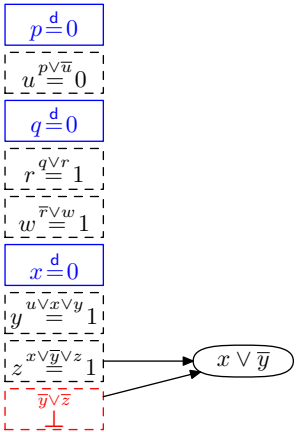
Could backtrack by erasing conflict level & flipping last decision (this is what DPLL does)

But want to learn from conflict and cut away as much of search space as possible

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Could backtrack by erasing conflict level & flipping last decision (this is what DPLL does)

But want to learn from conflict and cut away as much of search space as possible

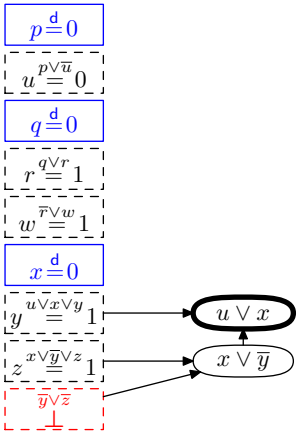
Case analysis for last two clauses over propagated variable:

- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Merge clauses & remove z — must satisfy $x \vee \overline{y}$

Conflict Analysis

Time to analyse this conflict and learn from it!

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Could backtrack by erasing conflict level & flipping last decision (this is what DPLL does)

But want to learn from conflict and cut away as much of search space as possible

Case analysis for last two clauses over propagated variable:

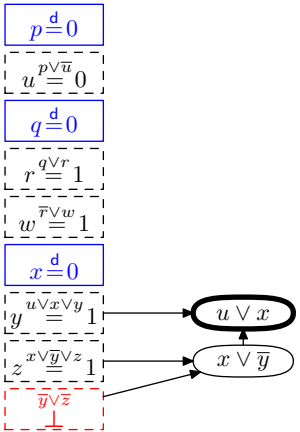
- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Merge clauses & remove z — must satisfy $x \vee \overline{y}$

Repeat until UIP clause with only 1 variable at conflict level after last decision — learn and backjump

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

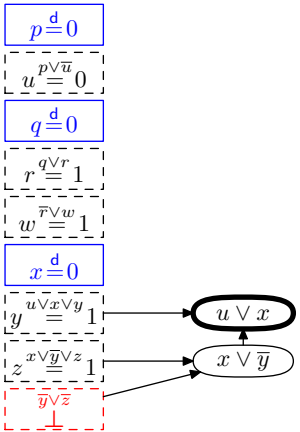
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



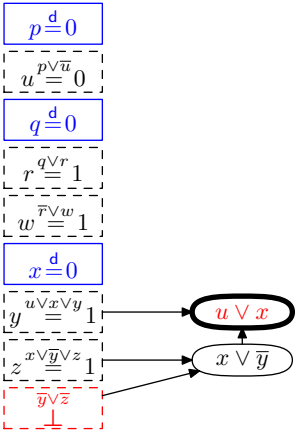
$p \overset{\mathsf{d}}{=} 0$
$u \overset{p \vee \overline{u}}{=} 0$

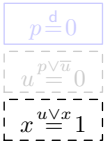
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$





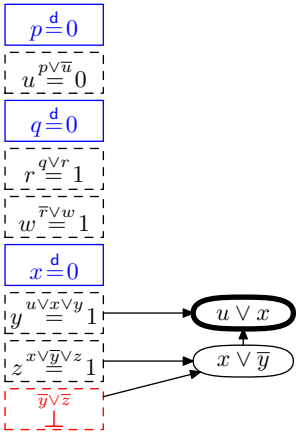
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

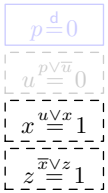
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$





Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

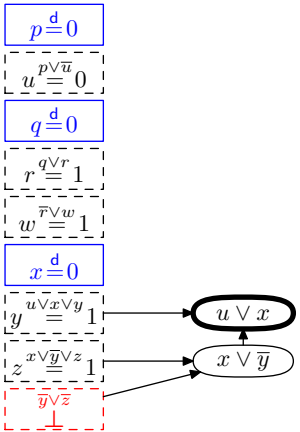
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

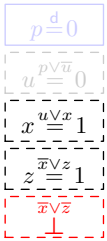
Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee u)$$

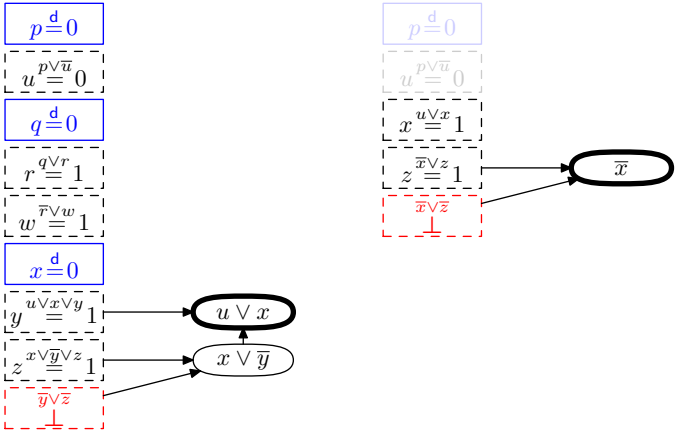




Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

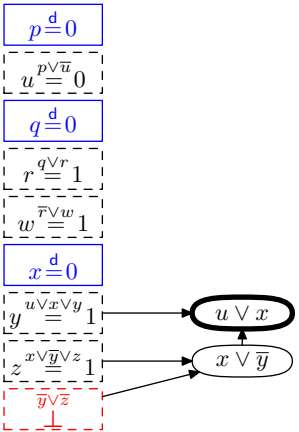
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee u)$$

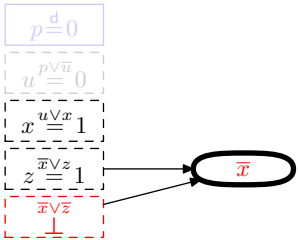


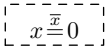
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$



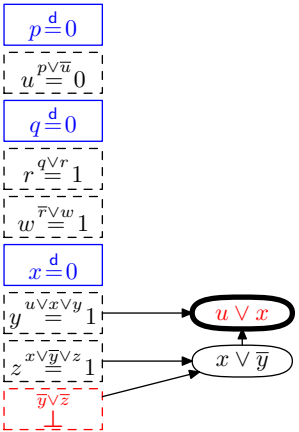


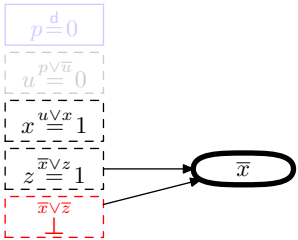


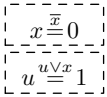
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee u)$$



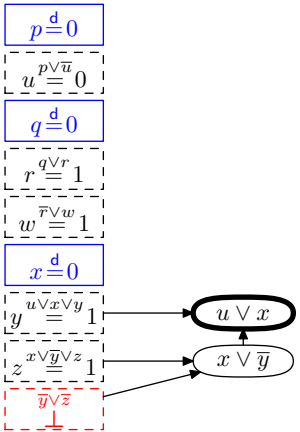


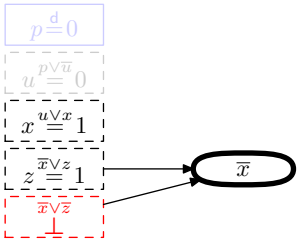


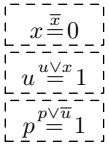
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee u)$



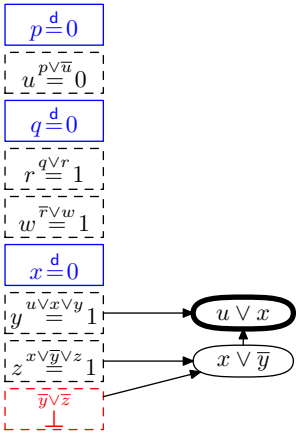


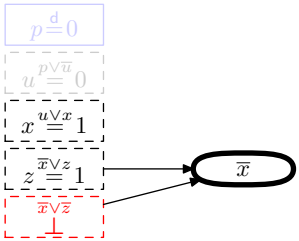


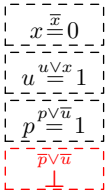
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



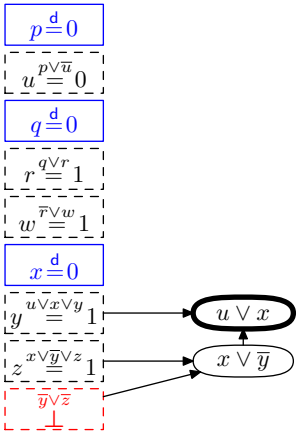


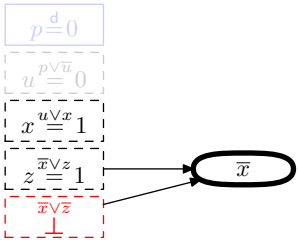


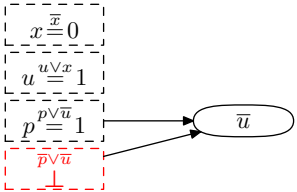
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



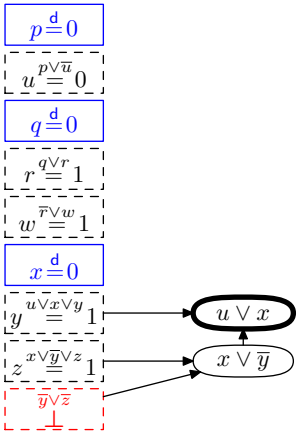


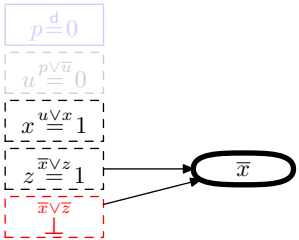


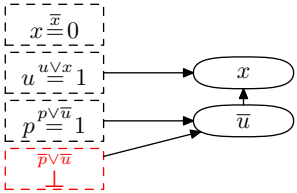
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



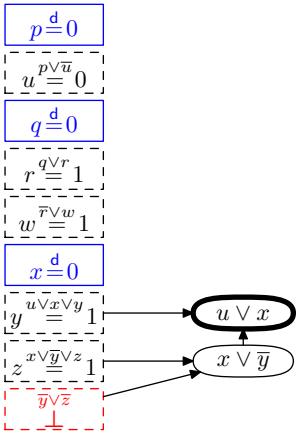


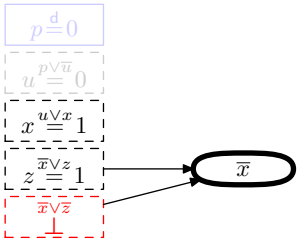


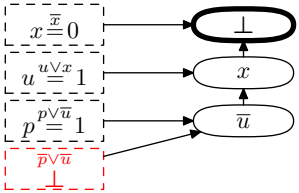
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$







Clause Database Reduction

- Solvers learn thousands of clauses per second
- In addition to learning clauses, also erase learned clauses that don't seem useful
- Modern solvers do this very aggressively
- Speeds up search (in particular, unit propagation, which dominates running time)

Restarts

- Fairly frequently, start search all over (but keep learned clauses)
- Original intuition: stuck in bad part of search tree — go somewhere else
- Not the reason this is done now
- Popular variables with high VSIDS scores get set again [MMZ+01]
- Are even set to same values (phase saving) [PD07]
- Current intuition: improves the search by focusing on important variables
- Restarts at fixed intervals or (better) adaptive restarts depending on "quality" of learned clauses
- Popular quality measure: literal block distance (LBD) = # decision levels represented in learned clause [AS09, AS12]

SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of CDCL SAT solver performance?
Many intricate, hard-to-understand heuristics
So focus instead on underlying method of reasoning

SAT Solver Analysis and the Resolution Proof System

How to make rigorous analysis of CDCL SAT solver performance?
Many intricate, hard-to-understand heuristics
So focus instead on underlying method of reasoning

Resolution proof system [Bla37, Rob65]

- Start with clauses of CNF formula (axioms)
- Derive new clauses by resolution rule

$$\frac{C_1 \vee x \qquad C_2 \vee \overline{x}}{C_1 \vee C_2}$$

Resolution Proofs by Contradiction

Resolution rule:

$$\frac{C_1 \vee x \qquad C_2 \vee \overline{x}}{C_1 \vee C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \wedge D$ is satisfiable.

Resolution Proofs by Contradiction

Resolution rule:

$$\frac{C_1 \vee x \qquad C_2 \vee \overline{x}}{C_1 \vee C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \wedge D$ is satisfiable.

So can prove F unsatisfiable by deriving the unsatisfiable empty clause (denoted $\bot$) from F by resolution

Resolution Proofs by Contradiction

Resolution rule:

$$\frac{C_1 \vee x \qquad C_2 \vee \overline{x}}{C_1 \vee C_2}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $D_1, D_2 \in F$ by the resolution rule, then $F \wedge D$ is satisfiable.

So can prove F unsatisfiable by deriving the unsatisfiable empty clause (denoted $\bot$) from F by resolution

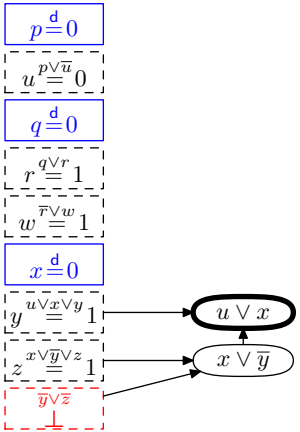
Such proof by contradiction also called resolution refutation

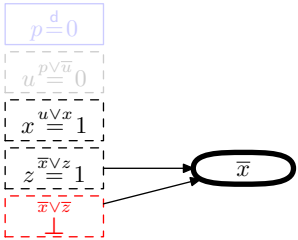
CDCL and Resolution Proofs

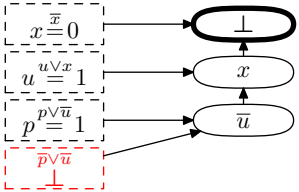
Obtain resolution proof...

CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution...

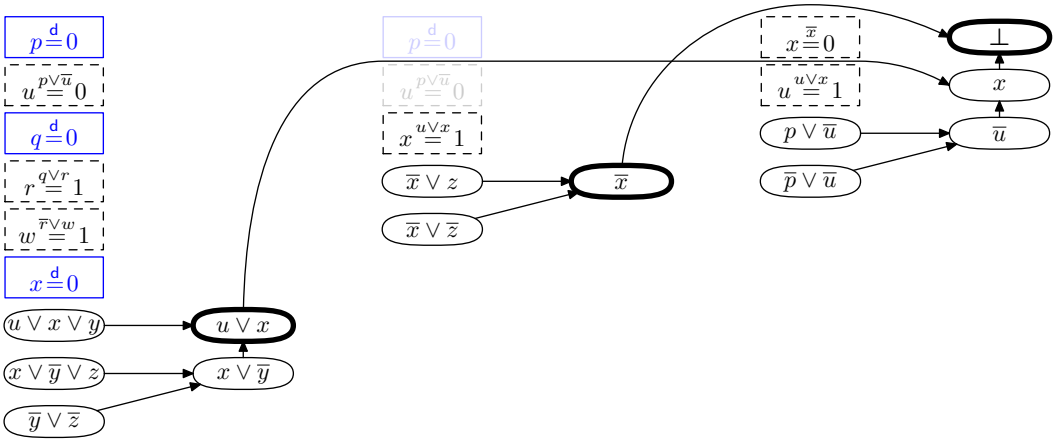






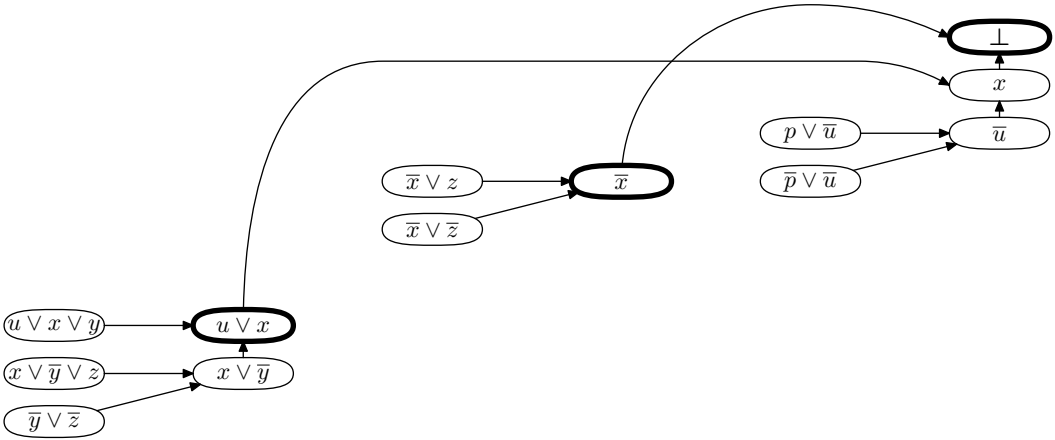
CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution by stringing together conflict analyses:



CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution
- Requires an argument, but you have seen enough to be able to fill in the needed details...

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution
- Requires an argument, but you have seen enough to be able to fill in the needed details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution
- Requires an argument, but you have seen enough to be able to fill in the needed details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$
  lower bounds on CDCL running time

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution
- Requires an argument, but you have seen enough to be able to fill in the needed details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$
  lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof from any CDCL execution
- Requires an argument, but you have seen enough to be able to fill in the needed details...
- This holds even for CDCL solvers with sophisticated heuristics and optimizations that we have not discussed*
- Hence, lower bounds on resolution proof size $\Rightarrow$
  lower bounds on CDCL running time
- Lower (and upper) bounds for different methods of reasoning about propositional logic formulas studied in proof complexity

(*) Except for some preprocessing techniques, which is an important omission, but this gets complicated and we don't have time to go into details...

CDCL and Efficient Proof Search

- Any CDCL solver reasoning (except preprocessing) can be captured by the resolution proof system
- But proofs resulting from CDCL execution have special structure, so not all resolution proofs can be generated
- Can CDCL always find resolution proofs efficiently? (With at most polynomial blow-up)

CDCL and Efficient Proof Search

- Any CDCL solver reasoning (except preprocessing) can be captured by the resolution proof system
- But proofs resulting from CDCL execution have special structure, so not all resolution proofs can be generated
- Can CDCL always find resolution proofs efficiently? (With at most polynomial blow-up)
- Very challenging question if CDCL is modelled faithfully,* but efficient proof search turns out to be possible assuming the right variable decisions [AFT11, PD11]

CDCL and Efficient Proof Search

- Any CDCL solver reasoning (except preprocessing) can be captured by the resolution proof system
- But proofs resulting from CDCL execution have special structure, so not all resolution proofs can be generated
- Can CDCL always find resolution proofs efficiently? (With at most polynomial blow-up)
- Very challenging question if CDCL is modelled faithfully,* but efficient proof search turns out to be possible assuming the right variable decisions [AFT11, PD11]
- However, turning this into a constructive algorithm is NP-hard [AM20]

CDCL and Efficient Proof Search

- Any CDCL solver reasoning (except preprocessing) can be captured by the resolution proof system
- But proofs resulting from CDCL execution have special structure, so not all resolution proofs can be generated
- Can CDCL always find resolution proofs efficiently? (With at most polynomial blow-up)
- Very challenging question if CDCL is modelled faithfully,* but efficient proof search turns out to be possible assuming the right variable decisions [AFT11, PD11]
- However, turning this into a constructive algorithm is NP-hard [AM20]

(*) The details again get a bit too complicated for the purposes of this talk, but a key technical difficulty is to handle that solvers always greedily propagate

The Unreasonable Effectiveness of SAT Solvers

- Modern CDCL SAT solvers solvers often perform amazingly well, and are used to solve instances with millions of variables on a daily basis
  ("SAT is easy in practice")

The Unreasonable Effectiveness of SAT Solvers

- Modern CDCL SAT solvers solvers often perform amazingly well, and are used to solve instances with millions of variables on a daily basis ("SAT is easy in practice")
- Lots of smart engineering to make it fly in practice [MMZ+01, ES04, EB05, PD07, Bie08, AS09, SB09, AS12, Oh16, ...]

The Unreasonable Effectiveness of SAT Solvers

- Modern CDCL SAT solvers solvers often perform amazingly well, and are used to solve instances with millions of variables on a daily basis ("SAT is easy in practice")
- Lots of smart engineering to make it fly in practice [MMZ+01, ES04, EB05, PD07, Bie08, AS09, SB09, AS12, Oh16, ...]
- ... And a somewhat bewildering alphabet soup of heuristics (VSIDS, 1UIP, LBD, BCD, BCE, BVA, BVE, ELS, FLP, VMTF, ...)

The Unreasonable Effectiveness of SAT Solvers

- Modern CDCL SAT solvers solvers often perform amazingly well, and are used to solve instances with millions of variables on a daily basis ("SAT is easy in practice")
- Lots of smart engineering to make it fly in practice [MMZ+01, ES04, EB05, PD07, Bie08, AS09, SB09, AS12, Oh16, ...]
- ... And a somewhat bewildering alphabet soup of heuristics (VSIDS, 1UIP, LBD, BCD, BCE, BVA, BVE, ELS, FLP, VMTF, ...)
- Very poor theoretical understanding:
  - Why and how do heuristics work?
  - Why are applied instances easy?

The Unreasonable Effectiveness of SAT Solvers

- Modern CDCL SAT solvers solvers often perform amazingly well, and are used to solve instances with millions of variables on a daily basis ("SAT is easy in practice")
- Lots of smart engineering to make it fly in practice [MMZ+01, ES04, EB05, PD07, Bie08, AS09, SB09, AS12, Oh16, ...]
- ... And a somewhat bewildering alphabet soup of heuristics (VSIDS, 1UIP, LBD, BCD, BCE, BVA, BVE, ELS, FLP, VMTF, ...)
- Very poor theoretical understanding:
  - Why and how do heuristics work?
  - Why are applied instances easy?
- Paradox: resolution quite weak proof system; many strong proof complexity lower bounds for (seemingly) "obvious" formulas encoding counting or parity arguments (e.g., [Hak85, Urq87, Pud97, BW01])

How to Analyse Performance of CDCL Solvers?

Applied approach

- Instrument solver with combinations of heuristics
- Run on SAT competition benchmarks
- Heterogeneous benchmarks
  - Poorly understood properties
  - Isolated data points
- Some work in [LM02, KSM11, SM11] — hard to draw conclusions

How to Analyse Performance of CDCL Solvers?

Applied approach

- Instrument solver with combinations of heuristics
- Run on SAT competition benchmarks
- Heterogeneous benchmarks
  - Poorly understood properties
  - Isolated data points
- Some work in [LM02, KSM11, SM11] — hard to draw conclusions

Theoretical approach

- Relate CDCL performance to complexity measures for resolution proof system?
- Only considers existence of proofs, not algorithmic search
- Only asymptotic results (sometimes for gigantic formulas)

Why Not Combine Theory and Practice?

Our combined approach using theoretical benchmark formulas [EGG$^+$18]

- Choose interesting formulas from proof complexity literature
  $\Rightarrow$ Well-understood properties

Why Not Combine Theory and Practice?

Our combined approach using theoretical benchmark formulas [EGG$^+$18]

- Choose interesting formulas from proof complexity literature
  $\Rightarrow$ Well-understood properties
- Tune to get theoretically easy benchmarks (but potentially tricky to solve in practice)
  $\Rightarrow$ Measure CDCL proof search quality

Why Not Combine Theory and Practice?

Our combined approach using theoretical benchmark formulas [EGG$^+$18]

- Choose interesting formulas from proof complexity literature
  $\Rightarrow$ Well-understood properties
- Tune to get theoretically easy benchmarks (but potentially tricky to solve in practice)
  $\Rightarrow$ Measure CDCL proof search quality
- Generate families with scaled instances of "same problem"
  $\Rightarrow$ Study not isolated data points but asymptotic behaviour

Why Not Combine Theory and Practice?

Our combined approach using theoretical benchmark formulas [EGG$^+$18]

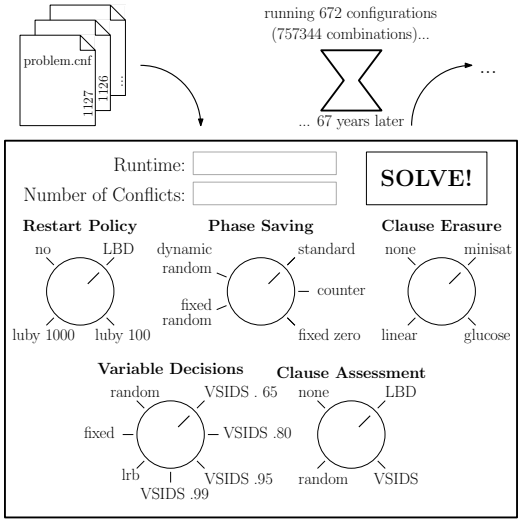
- Choose interesting formulas from proof complexity literature
  $\Rightarrow$ Well-understood properties
- Tune to get theoretically easy benchmarks (but potentially tricky to solve in practice)
  $\Rightarrow$ Measure CDCL proof search quality
- Generate families with scaled instances of "same problem"
  $\Rightarrow$ Study not isolated data points but asymptotic behaviour
- Focus on extremal benchmarks w.r.t. diverse properties
  $\Rightarrow$ Different benchmarks will "stress-test" different heuristics

Why Not Combine Theory and Practice?

Our combined approach using theoretical benchmark formulas [EGG$^+$18]

- Choose interesting formulas from proof complexity literature
  $\Rightarrow$ Well-understood properties
- Tune to get theoretically easy benchmarks (but potentially tricky to solve in practice)
  $\Rightarrow$ Measure CDCL proof search quality
- Generate families with scaled instances of "same problem"
  $\Rightarrow$ Study not isolated data points but asymptotic behaviour
- Focus on extremal benchmarks w.r.t. diverse properties
  $\Rightarrow$ Different benchmarks will "stress-test" different heuristics
- Code up instrumented solver with wide selection of heuristics
  - More complicated than it sounds — some heuristics tightly integrated
  - Check behaviour is as expected for settings mirroring *MiniSat* [Min] and *Glucose* [Glu]
  - Run on (essentially full) cross product of heuristics
  - Which heuristics are important when?
  - How do heuristics interact?

Our Set-up


problem.cnf
1127
1126
...
running 672 configurations
(757344 combinations)...
... 67 years later
...
Runtime:
Number of Conflicts:
SOLVE!
Restart Policy
no
LBD
luby 1000
luby 100
Phase Saving
dynamic
standard
random
counter
fixed
random
fixed zero
Clause Erasure
none
minisat
linear
glucose
Variable Decisions
random
VSIDS . 65
fixed
VSIDS .80
lrb
VSIDS .95
VSIDS .99
Clause Assessment
none
LBD
random
VSIDS

Some Possible Objections

- How do we know that we are observing proof-theoretic properties as opposed to other structural characteristics of the formulas?
- Especially since sometimes benchmarks have to be tweaked so that theoretical guarantees are lost...
- Are crafted benchmark formulas even relevant? — SAT solvers are designed to perform well on "real-world" benchmarks
- All benchmark formulas are unsatisfiable, but performance on satisfiable formulas is also important in practice
- Last but not least, correlation is not causation

Empirical Results

Experiments

- 1127 instances in 27 formula families
- 672 solver configurations
- More than 500,000 hours (67 years) of computations

Empirical Results

Experiments

- 1127 instances in 27 formula families
- 672 solver configurations
- More than 500,000 hours (67 years) of computations

Analysis?

- Huge amounts of data... How to even get an overview?
- How to make sure results are significant?
- Solvers deterministic — standard statistic tools don't apply

Empirical Results

Experiments

- 1127 instances in 27 formula families
- 672 solver configurations
- More than 500,000 hours (67 years) of computations

Analysis? (Topic for separate talk)

- Huge amounts of data... How to even get an overview?
- How to make sure results are significant?
- Solvers deterministic — standard statistic tools don't apply

Empirical Results

Experiments

- 1127 instances in 27 formula families
- 672 solver configurations
- More than 500,000 hours (67 years) of computations

Analysis? (Topic for separate talk)

- Huge amounts of data... How to even get an overview?
- How to make sure results are significant?
- Solvers deterministic — standard statistic tools don't apply

This talk: Focus on some example findings

- Clause learning
- Restarts
- Memory management
- Branching heuristic

Theory: Clause Learning and Tree-Like Resolution

- Search in CDCL solvers crucially guided by conflicts
- Clause learning influences
  - variable decisions
  - restart policy
  - memory management
  - and more...
- Is storing actual clauses learned during conflict analysis essential?

What theory says

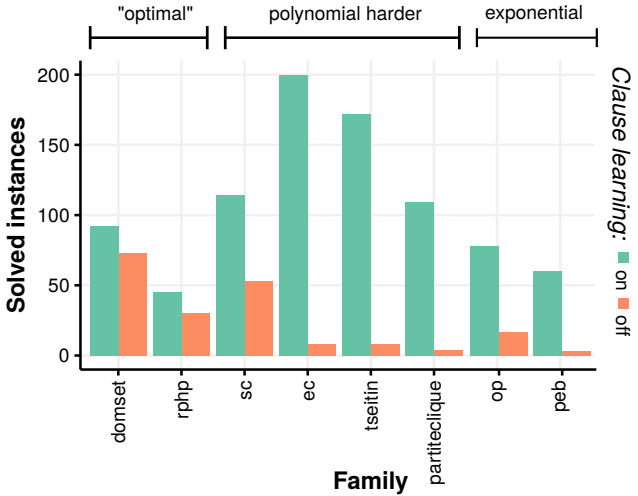
- no learning $\Rightarrow$ tree-like resolution (DPLL)

How to check in practice?

Run experiments on

- instances that separate tree-like and general resolution
- instances where tree-like is (asymptotically) optimal

Empirical: Clause Learning and Tree-Like Resolution



Restarts

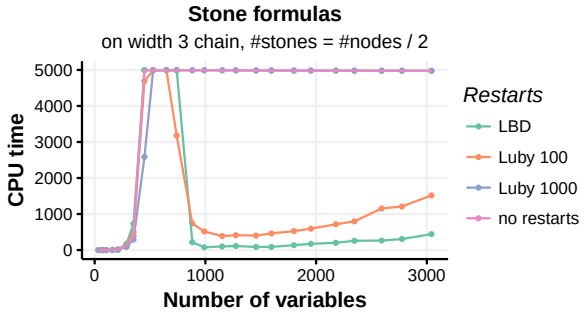
- Fast restarts crucial for CDCL solvers in practice
- Also key for showing that CDCL can find short resolution proofs [AFT11, PD11]
- But do restarts increase theoretical reasoning power? Open...

Restarts

- Fast restarts crucial for CDCL solvers in practice
- Also key for showing that CDCL can find short resolution proofs [AFT11, PD11]
- But do restarts increase theoretical reasoning power? Open...
- Run experiments on benchmarks where "full power of resolution" needed [AJPU07]
- Provides "circumstantial evidence" that restarts make solver reasoning stronger

Restarts

- Fast restarts crucial for CDCL solvers in practice
- Also key for showing that CDCL can find short resolution proofs [AFT11, PD11]
- But do restarts increase theoretical reasoning power? Open...
- Run experiments on benchmarks where "full power of resolution" needed [AJPU07]
- Provides "circumstantial evidence" that restarts make solver reasoning stronger

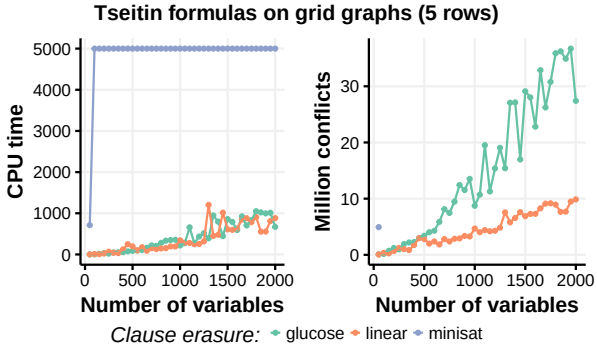


Memory Management: Time-Space Trade-offs

- CDCL solvers very aggressively minimize memory usage
- Dramatic time-space trade-offs in theory [BN11, BBI12, BNT13]
- Could such trade-off phenomena show up in practice? Seems like yes

Memory Management: Time-Space Trade-offs

- CDCL solvers very aggressively minimize memory usage
- Dramatic time-space trade-offs in theory [BN11, BBI12, BNT13]
- Could such trade-off phenomena show up in practice? Seems like yes



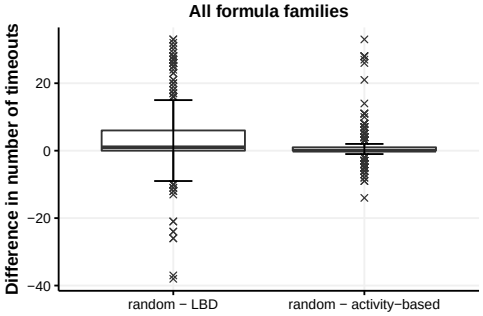
Database size: minisat $\left(\sim N^{0.25}\right)$ < glucose $\left(\sim N^{0.5}\right)$ < linear $\left(\sim N\right)$ [$N = \#\text{conflicts}$]

Memory Management: Clause Assessment

- When time to purge database, how to identify useful clauses to keep?
  - Activity-based [ES04]
  - Literal block distance (LBD) [AS09]
  - Random (control)

Memory Management: Clause Assessment

- When time to purge database, how to identify useful clauses to keep?
  - Activity-based [ES04]
  - Literal block distance (LBD) [AS09]
  - Random (control)
- LBD quite successful, especially when space is getting tight
- Activity-based indistinguishable from random (& random OK-ish)

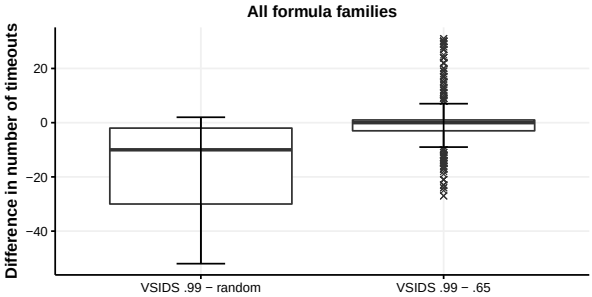


Branching Heuristics: Importance of Memory Horizon

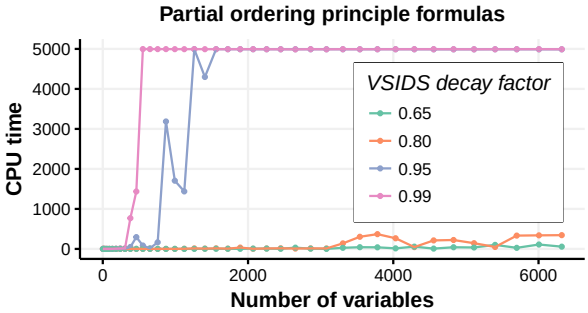
- Branch on variable appearing most often in recent conflicts
  + exponential decay to reward recent conflicts: VSIDS [MMZ$^+$01]
    - Low VSIDS factor = short memory
    - High VSIDS factor = long memory
- Choice high/low depends on type of formulas
- But random decisions are pretty much always bad

Branching Heuristics: Importance of Memory Horizon

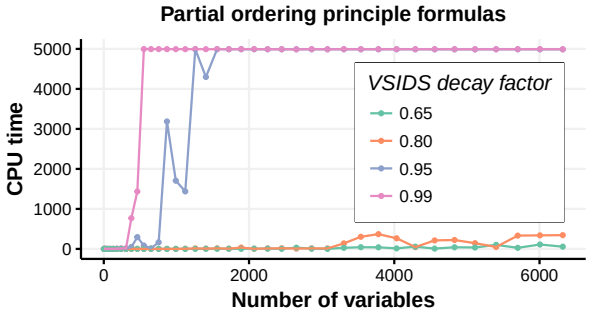
- Branch on variable appearing most often in recent conflicts
  + exponential decay to reward recent conflicts: VSIDS [MMZ+01]
    - Low VSIDS factor = short memory
    - High VSIDS factor = long memory
- Choice high/low depends on type of formulas
- But random decisions are pretty much always bad



Branching Heuristics: A Particularly Dramatic Example



Branching Heuristics: A Particularly Dramatic Example



- What's going on? We're not sure...
- Instances very hard for tree-like resolution ($\approx$ DPLL) [BG01, BW01]
- Low VSIDS factor $\Rightarrow$ search focus locally on latest conflicts
- Maybe high VSIDS factor $\Rightarrow$ closer to DPLL-style search?!
  (Consistent with our experiments, but more data & insights needed)

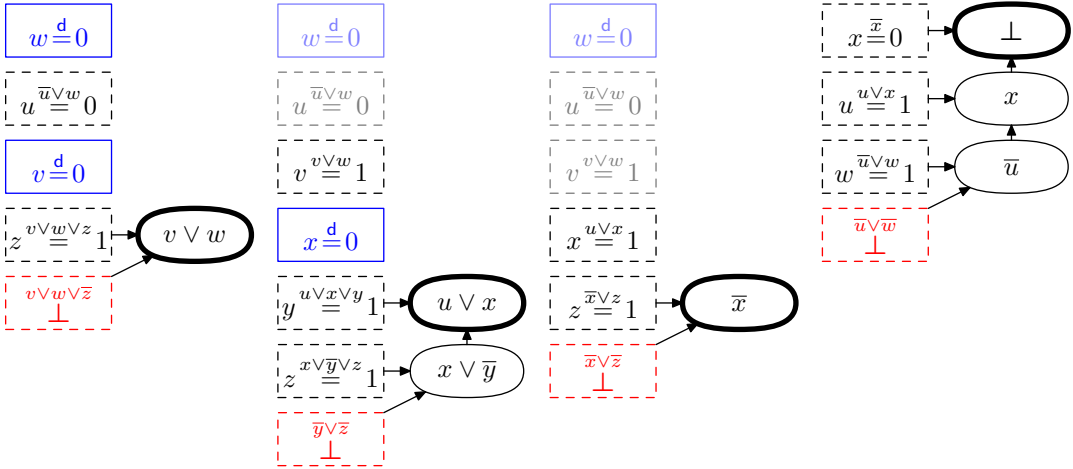
Second Approach: Study Solver-Generated Proofs

Analyze solver reasoning for applied benchmarks [KN20]
Measure impact of CDCL solver heuristics by

- Running SAT solvers with different settings on real-world benchmarks
- Looking at resolution proofs produced by solvers
- Smaller proof $\Rightarrow$ more efficient reasoning
- Also trim proofs to determine which clauses needed to reach final conclusion

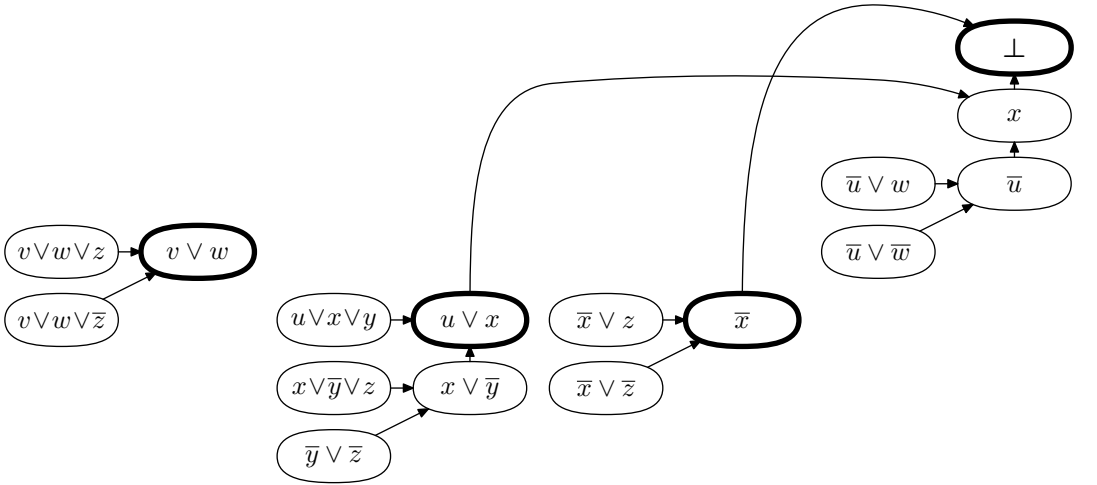
Proof Trimming by Picture

$$(u \vee x \vee y) \wedge (v \vee w \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (v \vee w \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$



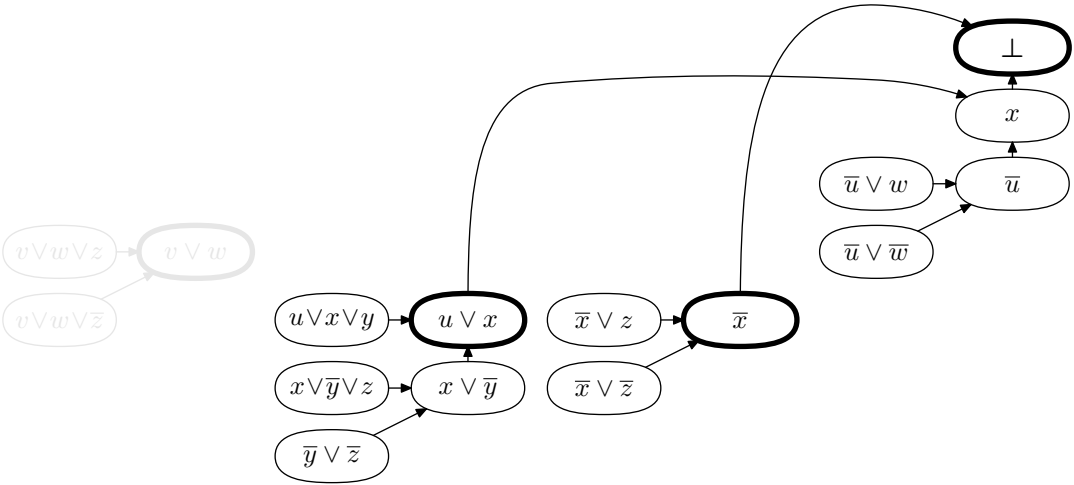
Proof Trimming by Picture

$$(u \vee x \vee y) \wedge (v \vee w \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (v \vee w \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$



Proof Trimming by Picture

$$(u \vee x \vee y) \wedge (v \vee w \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (v \vee w \vee \overline{z}) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$



Heuristics Studied

Restarts: Adaptive or non-adaptive

- Adaptive based on literal block distance (LBD)
- Fixed Luby restarts — multiples of sequence $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots)$

Heuristics Studied

Restarts: Adaptive or non-adaptive

- Adaptive based on literal block distance (LBD)
- Fixed Luby restarts — multiples of sequence $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots)$

Clause erasure: On/off

- Standard clause database reduction policy
- No clause erasures

Heuristics Studied

Restarts: Adaptive or non-adaptive

- Adaptive based on literal block distance (LBD)
- Fixed Luby restarts — multiples of sequence $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots)$

Clause erasure: On/off

- Standard clause database reduction policy
- No clause erasures

Clause assessment: Correlate usefulness of clauses with

- Literal block distance (LBD) score
- Conflict activity
- Clause size
- Decision level learned
- Backjump length caused
- # conflicts since last restart

Focus on clauses of size ≥ 3, since smaller clauses will never be removed

Benchmarks Selection

- Randomly sample 200 benchmarks from recent SAT competitions
- Then filter so that all relevant solver configurations can solve all benchmarks
- Concretely, for each experiment:
  - Let all configurations pick their 150 best benchmarks
  - Take intersection for all solver configurations used in experiment
  - Use whatever time it takes to get all solvers solve all these benchmarks
- In the end, 120–133 benchmarks per experiment
- And again fairly massive computations...

Analyzing Restarts Using Solver-Generated Proofs

Non-adaptive restarts (Luby sequence)

- Higher restart frequency makes both untrimmed and trimmed proofs smaller

Analyzing Restarts Using Solver-Generated Proofs

Non-adaptive restarts (Luby sequence)

- Higher restart frequency makes both untrimmed and trimmed proofs smaller

Adaptive restarts (based on LBD scores of learned clauses)

- Smaller untrimmed proofs than for non-adaptive restarts with same average restart frequency (so timing of restarts important)
- Not so significant size decrease for trimmed proofs
- Making restarts adaptive mainly helps avoid work not needed to reach the end result

Analyzing Restarts Using Solver-Generated Proofs

Non-adaptive restarts (Luby sequence)

- Higher restart frequency makes both untrimmed and trimmed proofs smaller

Adaptive restarts (based on LBD scores of learned clauses)

- Smaller untrimmed proofs than for non-adaptive restarts with same average restart frequency (so timing of restarts important)
- Not so significant size decrease for trimmed proofs
- Making restarts adaptive mainly helps avoid work not needed to reach the end result

General conclusion

- Restarts improve reasoning power of solvers

Analyzing Clause Erasure Using Solver-Generated Proofs

Effects of switching off clause erasure

- Often smaller untrimmed proofs (but not always!)
- Interestingly, no significant effect on trimmed proofs
- So core reasoning not stronger with more clauses in memory, but extra clauses help solver focus and avoid work that turns out to be useless with hindsight(?)

Analyzing Clause Assessment Using Solver-Generated Proofs

Which clauses are likely to contribute in proofs?

- Very low (good) LBD score excellent predictor
- Otherwise conflict activity is good indicator
- Random choice is clearly worse than both (difference from theory benchmarks)
- Other features measured did not have strong impact
- Lends more rigorous support for currently most popular clause database management heuristic [Oh16]

Clause Assessment in More Detail: Two types of experiments

1. **With clause deletions**
   - Fix non-adaptive strategy for # clauses to remove at database reduction time
   - Decide which clauses to erase based on LBD, conflict activity, or randomly
   - Findings:
     - ⋆ Proof size is smaller for LBD-based erasure than activity-based erasure
     - ⋆ Both are clearly better than random erasures
     - ⋆ But random is not a totally hopeless strategy

Clause Assessment in More Detail: Two types of experiments

1. **With clause deletions**
   - Fix non-adaptive strategy for # clauses to remove at database reduction time
   - Decide which clauses to erase based on LBD, conflict activity, or randomly
   - Findings:
     - ⋆ Proof size is smaller for LBD-based erasure than activity-based erasure
     - ⋆ Both are clearly better than random erasures
     - ⋆ But random is not a totally hopeless strategy

2. **Without clause deletions**
   - Study untrimmed and trimmed proofs (also done in [Sim14, MY16])
   - Clauses are "useful" if they remain in trimmed proof
   - Findings:
     - ⋆ Very good LBD score correlates with surviving trimming
     - ⋆ Conflict activity better predictor over broader range of values
     - ⋆ LBD better measure than than simply clause size
   - Agrees with clause database management heuristics proposed in [Oh16]

Looking Forward: Scope for Theoretical Research

Turn circumstantial evidence into hard mathematical proofs

- Determine whether restarts increase the theoretical reasoning power of SAT solvers
- Finds ways of formalizing other findings

Looking Forward: Scope for Theoretical Research

Turn circumstantial evidence into hard mathematical proofs

- Determine whether restarts increase the theoretical reasoning power of SAT solvers
- Finds ways of formalizing other findings

Explain why SAT is "easy in practice"

- Possible to give mathematical characterization of properties of "real-world problems"?
- Or to find interesting synthetic benchmarks that look like applied problems?
- Can one prove that CDCL SAT solvers run fast for interesting classes of formulas?

Looking Forward: Scope for Theoretical Research

Turn circumstantial evidence into hard mathematical proofs

- Determine whether restarts increase the theoretical reasoning power of SAT solvers
- Finds ways of formalizing other findings

Explain why SAT is "easy in practice"

- Possible to give mathematical characterization of properties of "real-world problems"?
- Or to find interesting synthetic benchmarks that look like applied problems?
- Can one prove that CDCL SAT solvers run fast for interesting classes of formulas?

Find proof systems that tightly characterize other solving paradigms

- MaxSAT and pseudo-Boolean solving
- 0–1 integer linear and mixed integer linear programming

Take-Away Message

This talk

- Goal: Not to improve SAT solvers, but to understand how the best solvers work
- SAT solvers are too hard to model, so run computational experiments
- Use crafted benchmark formulas to check if performance matches theory predictions
- Or analyze solver-generated proofs for "real-world" formulas

Take-Away Message

This talk

- Goal: Not to improve SAT solvers, but to understand how the best solvers work
- SAT solvers are too hard to model, so run computational experiments
- Use crafted benchmark formulas to check if performance matches theory predictions
- Or analyze solver-generated proofs for "real-world" formulas

Some possible directions for future research

- Analyze proofs for applied CNF formulas in more detail
- Extract and analyze "proofs" for satisfiable formulas(?)
- MaxSAT and pseudo-Boolean solvers
  - Study performance on crafted benchmarks
  - Analyze proofs for "real-world" formulas
  - Use results to improve solver heuristics (less well developed than for SAT)
- Use experimental results to formulate hypotheses for theoretical study

Take-Away Message

This talk

- Goal: Not to improve SAT solvers, but to understand how the best solvers work
- SAT solvers are too hard to model, so run computational experiments
- Use crafted benchmark formulas to check if performance matches theory predictions
- Or analyze solver-generated proofs for "real-world" formulas

Some possible directions for future research

- Analyze proofs for applied CNF formulas in more detail
- Extract and analyze "proofs" for satisfiable formulas(?)
- MaxSAT and pseudo-Boolean solvers
  - Study performance on crafted benchmarks
  - Analyze proofs for "real-world" formulas
  - Use results to improve solver heuristics (less well developed than for SAT)
- Use experimental results to formulate hypotheses for theoretical study

Thank you for your attention!

References I

[AFT11] Albert Atserias, Johannes Klaus Fichte, and Marc Thurley. Clause-learning algorithms with many restarts and bounded-width resolution. *Journal of Artificial Intelligence Research*, 40:353–373, January 2011. Preliminary version in *SAT '09*.

[AJPU07] Michael Alekhnovich, Jan Johannsen, Toniann Pitassi, and Alasdair Urquhart. An exponential separation between regular and general resolution. *Theory of Computing*, 3(5):81–102, May 2007. Preliminary version in *STOC '02*.

[AM20] Albert Atserias and Moritz Müller. Automating resolution is NP-hard. *Journal of the ACM*, 67(5):31:1–31:17, October 2020. Preliminary version in *FOCS '19*.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern SAT solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 399–404, July 2009.

[AS12] Gilles Audemard and Laurent Simon. Refining restarts strategies for SAT and UNSAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 118–126. Springer, October 2012.

References II

[AW13] Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In Michael Jünger and Gerhard Reinelt, editors, *Facets of Combinatorial Optimization*, pages 449–481. Springer, 2013.

[BBI12] Paul Beame, Chris Beck, and Russell Impagliazzo. Time-space tradeoffs in resolution: Superpolynomial lower bounds for superlinear space. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing (STOC '12)*, pages 213–232, May 2012.

[BG01] María Luisa Bonet and Nicola Galesi. Optimality of size-width tradeoffs for resolution. *Computational Complexity*, 10(4):261–276, December 2001. Preliminary version in *FOCS '99*.

[BHvMW21] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, February 2021.

[Bie08] Armin Biere. Adaptive restart strategies for conflict driven SAT solvers. In *Proceedings of the 11th International Conference on Theory and Applications of Satisfiability Testing (SAT '08)*, volume 4996 of *Lecture Notes in Computer Science*, pages 28–33. Springer, May 2008.

[Bla37] Archie Blake. *Canonical Expressions in Boolean Algebra*. PhD thesis, University of Chicago, 1937.

References III

[BN11] Eli Ben-Sasson and Jakob Nordström. Understanding space in proof complexity: Separations and trade-offs via substitutions. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS '11)*, pages 401–416, January 2011.

[BNT13] Chris Beck, Jakob Nordström, and Bangsheng Tang. Some trade-off results for polynomial calculus. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing (STOC '13)*, pages 813–822, May 2013.

[BR07] Robert Bixby and Edward Rothberg. Progress in computational mixed integer programming—A look back from the other side of the tipping point. *Annals of Operations Research*, 149(1):37–41, February 2007.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[BW01] Eli Ben-Sasson and Avi Wigderson. Short proofs are narrow—resolution made simple. *Journal of the ACM*, 48(2):149–169, March 2001. Preliminary version in *STOC '99*.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

References IV

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[EB05] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT '05)*, volume 3569 of *Lecture Notes in Computer Science*, pages 61–75. Springer, June 2005.

[EGG+18] Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström, and Laurent Simon. Seeking practical CDCL insights from theoretical SAT benchmarks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1300–1308, July 2018.

[EGNV18] Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, and Marc Vinyals. Using combinatorial benchmarks to probe the reasoning power of pseudo-Boolean solvers. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 75–93. Springer, July 2018.

[ES04] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In *6th International Conference on Theory and Applications of Satisfiability Testing (SAT '03), Selected Revised Papers*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2004.

[Glu] The Glucose SAT solver. `https://www.labri.fr/perso/lsimon/research/glucose/`.

References V

[Hak85] Armin Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[JMNŽ12] Matti Järvisalo, Arie Matsliah, Jakob Nordström, and Stanislav Živný. Relating proof complexity measures and practical hardness of SAT. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming (CP '12)*, volume 7514 of *Lecture Notes in Computer Science*, pages 316–331. Springer, October 2012.

[KN20] Janne I. Kokkala and Jakob Nordström. Using resolution proofs to analyse CDCL solvers. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 427–444. Springer, September 2020.

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *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, June 2011.

[LM02] Inês Lynce and João P. Marques-Silva. Building state-of-the-art SAT solvers. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI '02)*, volume 77 of *Frontiers in Artificial Intelligence and Applications*, pages 166–170, May 2002.

References VI

[Min] The MiniSat page. `http://minisat.se/`.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC '01)*, pages 530–535, June 2001.

[MS99] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, May 1999. Preliminary version in *ICCAD '96*.

[MY16] Sharad Malik and Victor A. Ying. On the efficiency of the VSIDS decision heuristic. Presentation at the workshop *Theoretical Foundations of SAT Solving*. Slides available at `http://www.fields.utoronto.ca/sites/default/files/talk-attachments/SharadMalik-Fields2016.pdf`, August 2016.

[Oh16] Chanseok Oh. *Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL*. PhD thesis, New York University, 2016. Available at `https://cs.nyu.edu/media/publications/oh_chanseok.pdf`.

References VII

[PD07] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing (SAT '07)*, volume 4501 of *Lecture Notes in Computer Science*, pages 294–299. Springer, May 2007.

[PD11] Knot Pipatsrisawat and Adnan Darwiche. On the power of clause-learning SAT solvers as resolution engines. *Artificial Intelligence*, 175(2):512–525, February 2011. Preliminary version in *CP '09*.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *Journal of Symbolic Logic*, 62(3):981–998, September 1997.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[SB09] Niklas Sörensson and Armin Biere. Minimizing learned clauses. In *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, July 2009.

References VIII

[Sim14] Laurent Simon. Post mortem analysis of SAT solver proofs. In *Proceedings of the 5th Pragmatics of SAT workshop*, volume 27 of *EPiC Series in Computing*, pages 26–40, July 2014. Available at `https://easychair.org/publications/paper/N3GD`.

[SM11] Karem A. Sakallah and João Marques-Silva. Anatomy and empirical evaluation of modern SAT solvers. *Bulletin of the European Association for Theoretical Computer Science*, 103:96–121, February 2011.

[SS77] Richard M. Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, October 1977.

[Urq87] Alasdair Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, January 1987.