Conflict-Driven Solving for Boolean Satisfiability and Beyond

Jakob Nordström

University of Copenhagen and Lund University

Mathematics and Algorithms for Intelligent Decision-Making Summer Seminar 2026
Linköping University
June 8, 2026

This Is Me...

Jakob Nordström
Professor
University of Copenhagen
(with side affiliation at Lund University)
www.jakobnordstrom.se

This Is Me...

Jakob Nordström
Professor
University of Copenhagen
(with side affiliation at Lund University)
`www.jakobnordstrom.se`

Academic path:

- PhD at KTH Royal Institute of Technology 2008
- Postdoc at MIT 2008-2010
- Faculty at KTH 2011-2019
- In Copenhagen and Lund since 2019

. . . And This Is What I Do for a Living

$$(x_{1,1} \vee x_{1,2} \vee x_{1,3} \vee x_{1,4} \vee x_{1,5} \vee x_{1,6} \vee x_{1,7}) \wedge (x_{2,1} \vee x_{2,2} \vee x_{2,3} \vee x_{2,4} \vee x_{2,5} \vee x_{2,6} \vee x_{2,7}) \wedge (x_{3,1} \vee x_{3,2} \vee x_{3,3} \vee x_{3,4} \vee x_{3,5} \vee x_{3,6} \vee x_{3,7}) \wedge (x_{4,1} \vee x_{4,2} \vee x_{4,3} \vee x_{4,4} \vee x_{4,5} \vee x_{4,6} \vee x_{4,7}) \wedge (x_{5,1} \vee x_{5,2} \vee x_{5,3} \vee x_{5,4} \vee x_{5,5} \vee x_{5,6} \vee x_{5,7}) \wedge (x_{6,1} \vee x_{6,2} \vee x_{6,3} \vee x_{6,4} \vee x_{6,5} \vee x_{6,6} \vee x_{6,7}) \wedge (x_{7,1} \vee x_{7,2} \vee x_{7,3} \vee x_{7,4} \vee x_{7,5} \vee x_{7,6} \vee x_{7,7}) \wedge (x_{8,1} \vee x_{8,2} \vee x_{8,3} \vee x_{8,4} \vee x_{8,5} \vee x_{8,6} \vee x_{8,7}) \wedge (\neg x_{1,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,1} \vee \neg x_{3,1}) \wedge (\neg x_{1,1} \vee \neg x_{4,1}) \wedge (\neg x_{1,1} \vee \neg x_{5,1}) \wedge (\neg x_{1,1} \vee \neg x_{6,1}) \wedge (\neg x_{1,1} \vee \neg x_{7,1}) \wedge (\neg x_{1,1} \vee \neg x_{8,1}) \wedge (\neg x_{2,1} \vee \neg x_{3,1}) \wedge (\neg x_{2,1} \vee \neg x_{4,1}) \wedge (\neg x_{2,1} \vee \neg x_{5,1}) \wedge (\neg x_{2,1} \vee \neg x_{6,1}) \wedge (\neg x_{2,1} \vee \neg x_{7,1}) \wedge (\neg x_{2,1} \vee \neg x_{8,1}) \wedge (\neg x_{3,1} \vee \neg x_{4,1}) \wedge (\neg x_{3,1} \vee \neg x_{5,1}) \wedge (\neg x_{3,1} \vee \neg x_{6,1}) \wedge (\neg x_{3,1} \vee \neg x_{7,1}) \wedge (\neg x_{3,1} \vee \neg x_{8,1}) \wedge (\neg x_{4,1} \vee \neg x_{5,1}) \wedge (\neg x_{4,1} \vee \neg x_{6,1}) \wedge (\neg x_{4,1} \vee \neg x_{7,1}) \wedge (\neg x_{4,1} \vee \neg x_{8,1}) \wedge (\neg x_{5,1} \vee \neg x_{6,1}) \wedge (\neg x_{5,1} \vee \neg x_{7,1}) \wedge (\neg x_{5,1} \vee \neg x_{8,1}) \wedge (\neg x_{6,1} \vee \neg x_{7,1}) \wedge (\neg x_{6,1} \vee \neg x_{8,1}) \wedge (\neg x_{7,1} \vee \neg x_{8,1}) \wedge (\neg x_{1,2} \vee \neg x_{2,2}) \wedge (\neg x_{1,2} \vee \neg x_{3,2}) \wedge (\neg x_{1,2} \vee \neg x_{4,2}) \wedge (\neg x_{1,2} \vee \neg x_{5,2}) \wedge (\neg x_{1,2} \vee \neg x_{6,2}) \wedge (\neg x_{1,2} \vee \neg x_{7,2}) \wedge (\neg x_{1,2} \vee \neg x_{8,2}) \wedge (\neg x_{2,2} \vee \neg x_{3,2}) \wedge (\neg x_{2,2} \vee \neg x_{4,2}) \wedge (\neg x_{2,2} \vee \neg x_{5,2}) \wedge (\neg x_{2,2} \vee \neg x_{6,2}) \wedge (\neg x_{2,2} \vee \neg x_{7,2}) \wedge (\neg x_{2,2} \vee \neg x_{8,2}) \wedge (\neg x_{3,2} \vee \neg x_{4,2}) \wedge (\neg x_{3,2} \vee \neg x_{5,2}) \wedge (\neg x_{3,2} \vee \neg x_{6,2}) \wedge (\neg x_{3,2} \vee \neg x_{7,2}) \wedge (\neg x_{3,2} \vee \neg x_{8,2}) \wedge (\neg x_{4,2} \vee \neg x_{5,2}) \wedge (\neg x_{4,2} \vee \neg x_{6,2}) \wedge (\neg x_{4,2} \vee \neg x_{7,2}) \wedge (\neg x_{4,2} \vee \neg x_{8,2}) \wedge (\neg x_{5,2} \vee \neg x_{6,2}) \wedge (\neg x_{5,2} \vee \neg x_{7,2}) \wedge (\neg x_{5,2} \vee \neg x_{8,2}) \wedge (\neg x_{6,2} \vee \neg x_{7,2}) \wedge (\neg x_{6,2} \vee \neg x_{8,2}) \wedge (\neg x_{7,2} \vee \neg x_{8,2}) \wedge (\neg x_{1,3} \vee \neg x_{2,3}) \wedge (\neg x_{1,3} \vee \neg x_{3,3}) \wedge (\neg x_{1,3} \vee \neg x_{4,3}) \wedge (\neg x_{1,3} \vee \neg x_{5,3}) \wedge (\neg x_{1,3} \vee \neg x_{6,3}) \wedge (\neg x_{1,3} \vee \neg x_{7,3}) \wedge (\neg x_{1,3} \vee \neg x_{8,3}) \wedge (\neg x_{2,3} \vee \neg x_{3,3}) \wedge (\neg x_{2,3} \vee \neg x_{4,3}) \wedge (\neg x_{2,3} \vee \neg x_{5,3}) \wedge (\neg x_{2,3} \vee \neg x_{6,3}) \wedge (\neg x_{2,3} \vee \neg x_{7,3}) \wedge (\neg x_{2,3} \vee \neg x_{8,3}) \wedge (\neg x_{3,3} \vee \neg x_{4,3}) \wedge (\neg x_{3,3} \vee \neg x_{5,3}) \wedge (\neg x_{3,3} \vee \neg x_{6,3}) \wedge (\neg x_{3,3} \vee \neg x_{7,3}) \wedge (\neg x_{3,3} \vee \neg x_{8,3}) \wedge (\neg x_{4,3} \vee \neg x_{5,3}) \wedge (\neg x_{4,3} \vee \neg x_{6,3}) \wedge (\neg x_{4,3} \vee \neg x_{7,3}) \wedge (\neg x_{4,3} \vee \neg x_{8,3}) \wedge (\neg x_{5,3} \vee \neg x_{6,3}) \wedge (\neg x_{5,3} \vee \neg x_{7,3}) \wedge (\neg x_{5,3} \vee \neg x_{8,3}) \wedge$$

In a Bit More Detail...

$$(x \vee z) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee u) \wedge (\neg y \vee \neg u)$$
$$\wedge\ (u \vee v) \wedge (\neg x \vee \neg v) \wedge (\neg u \vee w) \wedge (\neg x \vee \neg u \vee \neg w)$$

In a Bit More Detail...

$$(x \vee z) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee u) \wedge (\neg y \vee \neg u)$$
$$\wedge\ (u \vee v) \wedge (\neg x \vee \neg v) \wedge (\neg u \vee w) \wedge (\neg x \vee \neg u \vee \neg w)$$

- Variables should be set to **true** $(= 1)$ or **false** $(= 0)$
- Constraints like $(x \vee \neg y \vee u)$ means x or u should be true or y false
- $\wedge$ means all constraints should hold simultaneously
- Is there a truth value assignment satisfying all constraints?

In a Bit More Detail...

$$(x \vee z) \wedge (y \vee \neg z) \wedge (x \vee \neg y \vee u) \wedge (\neg y \vee \neg u)$$
$$\wedge\ (u \vee v) \wedge (\neg x \vee \neg v) \wedge (\neg u \vee w) \wedge (\neg x \vee \neg u \vee \neg w)$$

- Variables should be set to **true** $(= 1)$ or **false** $(= 0)$
- Constraints like $(x \vee \neg y \vee u)$ means x or u should be true or y false
- $\wedge$ means all constraints should hold simultaneously
- Is there a truth value assignment satisfying all constraints?

For operations research listeners:

- Think of $x \vee \neg y \vee u$ as $x + (1 - y) + u \geq 1$
- Just a special type of 0–1 linear program (without objective)

Some Highly Concrete Applications of Such Very Abstract Problems

- Software analysis, testing, and synthesis [DMB11]
- Hardware verification [Sha09]
- Air and train traffic control [ABFP12, FFH+16, ZR14]
- Smart crypto contracts [AGRS20, AGH+22]
- Gene regulatory network inference [PBD+22]
- Computational protein design [AAB+14, HD19]
- Assigning donated organs for transplants [MO12, BvdKM+21]
- Allocation of education and work opportunities [Man16, MMT17]
- Proving theorems in pure mathematics [HK17]

Some Highly Concrete Applications of Such Very Abstract Problems

- Software analysis, testing, and synthesis [DMB11]
- Hardware verification [Sha09]
- Air and train traffic control [ABFP12, FFH+16, ZR14]
- Smart crypto contracts [AGRS20, AGH+22]
- Gene regulatory network inference [PBD+22]
- Computational protein design [AAB+14, HD19]
- Assigning donated organs for transplants [MO12, BvdKM+21]
- Allocation of education and work opportunities [Man16, MMT17]
- Proving theorems in pure mathematics [HK17]

(Requires fleshing out quite a bit of details, but in the interest of time we will take this on trust. . .)

A Deep, Challenging Problem...

- Real-world problems correspond to **humongous** formulas (with 100,000s or even 1,000,000s of variables)
- Can we use computers to solve these problems efficiently?
- Question mentioned already in Gödel's famous letter in 1956 to von Neumann (the "father of computer science")
- Topic of intense research in computer science ever since 1960s
- Problem known to be computationally very challenging (NP-complete or worse) [Coo71, Lev73]
- Widely believed to be exponentially hard in the worst case [IP01, CIP09]

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?
  - How can they be so good in practice?

. . . For Which Amazing Applied Advances Have Been Made

- Enormous progress since the turn of the millennium on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But. . . There are also small formulas (just $\sim$100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known SAT solving algorithms based on Davis-Putnam-Logemann-Loveland or DPLL method from early 1960s (although with many clever optimizations)
- Some natural questions:
  - How do these SAT solvers work?
  - How can they be so good in practice?
  - When they fail to be efficient, can we understand why?

Plan for the Rest of This Talk

What we will cover in this survey:

Plan for the Rest of This Talk

What we will cover in this survey:

- Define more precisely the computational problem

Plan for the Rest of This Talk

What we will cover in this survey:

- Define more precisely the computational problem
- Give (slightly simplified) description of how modern SAT solvers work

Plan for the Rest of This Talk

What we will cover in this survey:

- Define more precisely the computational problem
- Give (slightly simplified) description of how modern SAT solvers work
- See how tools from computational complexity theory can be used to analyze performance of SAT solvers

Plan for the Rest of This Talk

What we will cover in this survey:

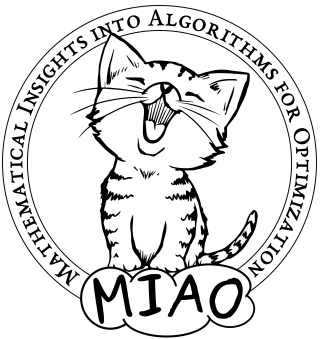
- Define more precisely the computational problem
- Give (slightly simplified) description of how modern SAT solvers work
- See how tools from computational complexity theory can be used to analyze performance of SAT solvers
- Discuss how to extend SAT solving techniques to stronger combinatorial solving paradigms

Plan for the Rest of This Talk

What we will cover in this survey:

- Define more precisely the computational problem
- Give (slightly simplified) description of how modern SAT solvers work
- See how tools from computational complexity theory can be used to analyze performance of SAT solvers
- Discuss how to extend SAT solving techniques to stronger combinatorial solving paradigms

. . . And in the process also touch on some of the research conducted in the Mathematical Insights into Algorithms for Optimization (MIAO) group in Copenhagen and Lund



Outline of Survey on Conflict-Driven Solving for SAT and Beyond

1. Boolean Satisfiability (SAT) Solving
   - Davis-Putnam-Logemann-Loveland (DPLL) Method
   - Conflict-Driven Clause Learning (CDCL)
   - Analyzing SAT Solvers Using Proof Complexity
2. Conflict-Driven Search for Stronger Combinatorial Solving Paradigms
   - Mixed Integer Programming (MIP) and Constraint Programming (CP)
   - Conflict Analysis Beyond SAT
   - Opportunities and Challenges
3. Research Outlook
   - Research in the MIAO Group
   - Pointers for Further Study

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

The SATISFIABILITY (or just SAT) Problem

Given a CNF formula F, is it satisfiable?

Formal Description of SAT Problem

- Variable x: takes value 1 (**true**) or 0 (**false**)
- Literal ℓ: variable x or its negation $\overline{x}$ (write $\overline{x}$ instead of $\neg x$)
- Clause $C = \ell_1 \vee \cdots \vee \ell_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- Conjunctive normal form (CNF) formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

The SATISFIABILITY (or just SAT) Problem

Given a CNF formula F, is it satisfiable?

For instance, what about our example formula?

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

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call
- Set $x = 1$, simplify F and make recursive call

Solving SAT by Case Analysis: DPLL

So the SAT problem is NP-complete and maybe exponentially hard, but what do you do if you have to solve it anyway?

Chances are you'll use some variant of the DPLL method developed by Davis, Putnam, Logemann & Loveland [DP60, DLL62]

DPLL (somewhat simplified description)

- If F contains empty clause (without literals), report "unsatisfiable" and return — refer to as conflict
- If F contains no clauses, report "satisfiable" and terminate
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and make recursive call
- Set $x = 1$, simplify F and make recursive call
- If result in both cases "unsatisfiable", then report "unsatisfiable" and return

A DPLL Toy Example

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

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals

A DPLL Toy Example

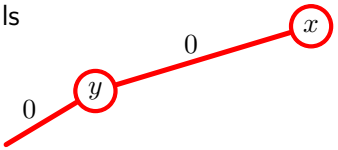
$$(\quad z) \wedge (\quad \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

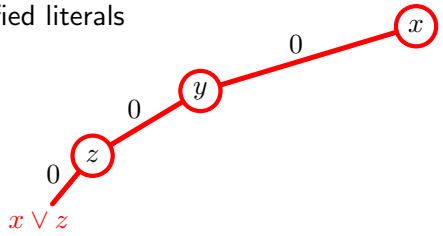
$$(x \vee z) \wedge (\quad \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

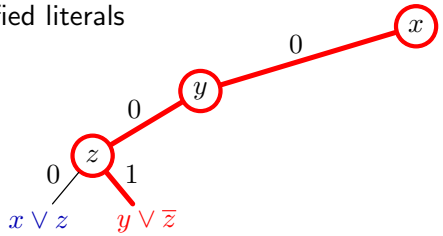
$$(\quad z) \wedge (y \vee \overline{z}) \wedge (\quad \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

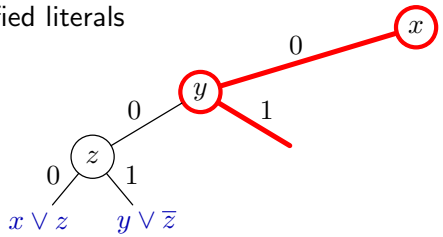
$$(\quad z) \wedge (y \vee \overline{z}) \wedge (\qquad u) \wedge (\quad \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

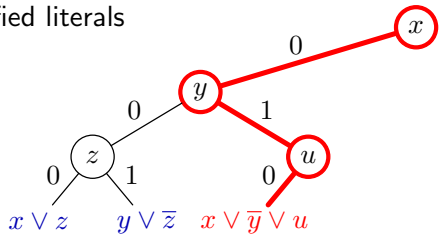
$$(\quad z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\quad \overline{u})$$
$$\wedge (\quad v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

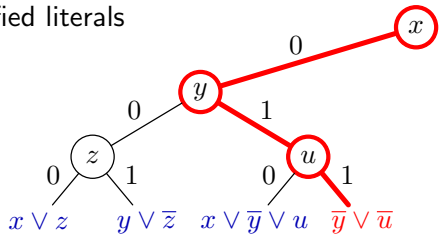
$$(\quad z) \wedge (y \vee \overline{z}) \wedge (\qquad u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge\ (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\quad w) \wedge (\overline{x} \vee \qquad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

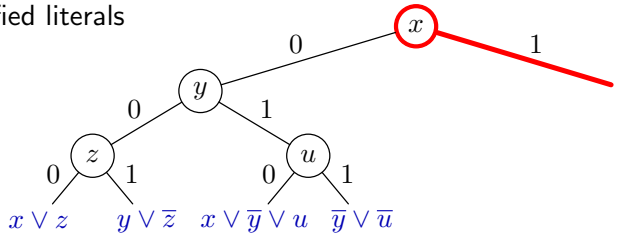
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\quad\ \overline{v}) \wedge (\overline{u} \vee w) \wedge (\quad\ \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

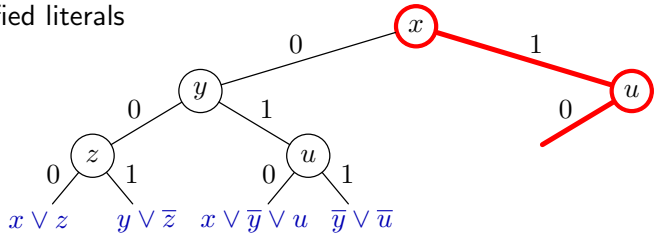
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \quad) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (\quad v) \wedge (\quad \overline{v}) \wedge (\overline{u} \vee w) \wedge (\quad \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

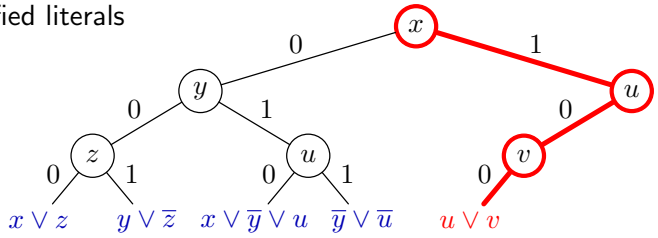
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y}) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

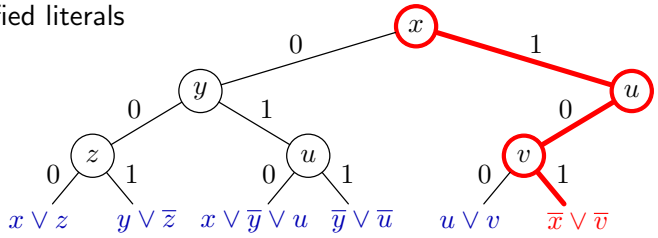
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \quad\quad) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (\quad\quad v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\quad\quad \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

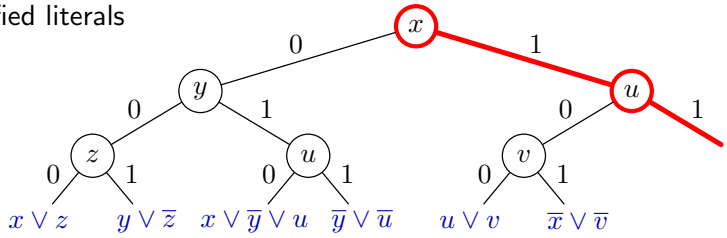
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\quad \overline{v}) \wedge (\quad w) \wedge (\quad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

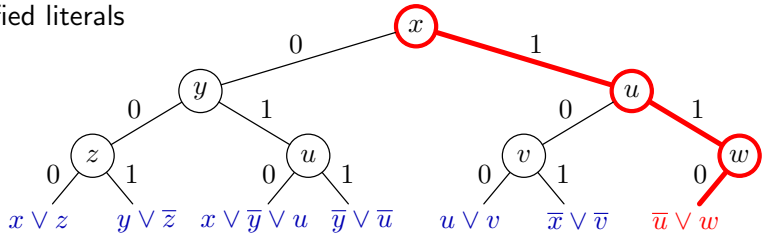
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\quad\ \overline{v}) \wedge (\overline{u} \vee w) \wedge (\qquad\quad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

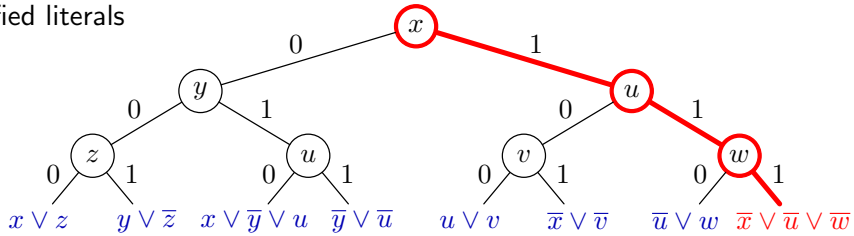
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\quad \overline{v}) \wedge (\quad w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



A DPLL Toy Example

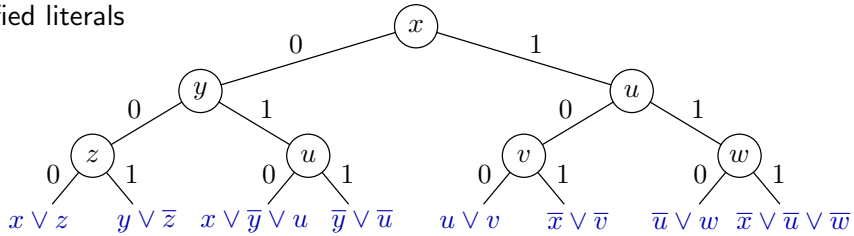
$$(x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u})$$
$$\wedge (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when conflict reached

"Simplify formula" by (mentally) removing

- satisfied clauses
- falsified literals



State-of-the-art SAT Solvers: Ingredients

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 reaching leaf, 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

Let us discuss some of these ingredients

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 aim for ≈99% of assignments being 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 aim for ≈99% of assignments being 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 variable with highest score

Clause Learning

- At conflict, want to add clause avoiding same part of search tree being explored again

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

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])

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]

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]
- Often described in terms of cuts in conflict graph

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]
- Often described in terms of cuts in conflict graph
- More helpful to view conflict analysis as syntactic derivation applied on 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:

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

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

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

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:

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

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

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

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

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)

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

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

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)

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

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

$y \overset{u \vee x \vee y}{=} 1$

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

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

$y \overset{u \vee x \vee y}{=} 1$

$z \overset{x \vee \overline{y} \vee z}{=} 1$

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:

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

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

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

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

$r \overset{q \vee r}{=} 1$

$w \overset{\overline{r} \vee w}{=} 1$

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

$y \overset{u \vee x \vee y}{=} 1$

$z \overset{x \vee \overline{y} \vee z}{=} 1$

$\overline{y} \vee \overline{z}$

$\bot$

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

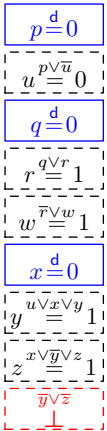
Add to assignment trail

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

Conflict Analysis

Time to analyse this conflict and learn from it!

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

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

decision level 1

$q \overset{\mathsf{d}}{=} 0$
$r \overset{q \vee r}{=} 1$
$w \overset{\overline{r} \vee w}{=} 1$

decision level 2

$x \overset{\mathsf{d}}{=} 0$
$y \overset{u \vee x \vee y}{=} 1$
$z \overset{x \vee \overline{y} \vee z}{=} 1$
$\overline{y} \vee \overline{z}$
$\bot$

decision level 3

Conflict Analysis

Time to analyse this conflict and learn from it!

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

$p \overset{\mathsf{d}}{=} 0$ — decision level 1

$u \overset{p \vee \overline{u}}{=} 0$ — decision level 1

$q \overset{\mathsf{d}}{=} 0$ — decision level 2

$r \overset{q \vee r}{=} 1$ — decision level 2

$w \overset{\overline{r} \vee w}{=} 1$ — decision level 2

$x \overset{\mathsf{d}}{=} 0$ — decision level 3

$y \overset{u \vee x \vee y}{=} 1$ — decision level 3

$z \overset{x \vee \overline{y} \vee z}{=} 1$ — decision level 3

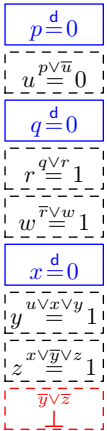
$\overset{\overline{y} \vee \overline{z}}{\bot}$ — decision level 3

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!

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



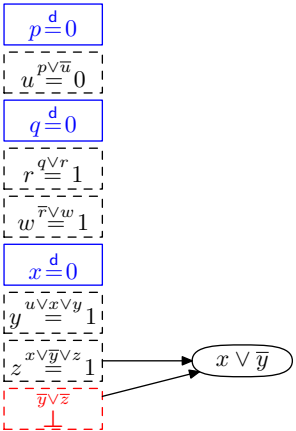
Could backtrack by erasing conflict level & flipping last decision (this is what DPLL does)

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

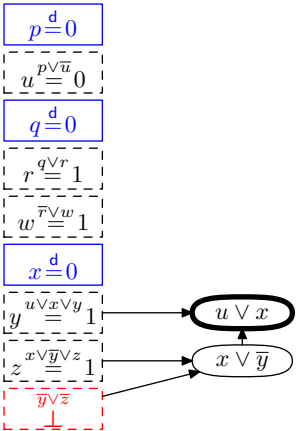
Case analysis for last two clauses over propagated variable:

- $x \vee \overline{y} \vee z$ 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:

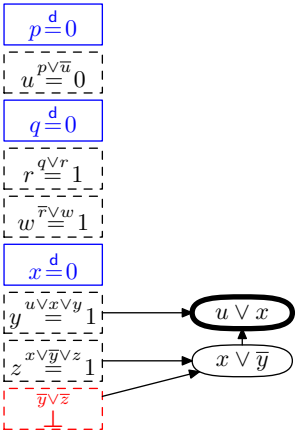
- $x \vee \overline{y} \vee z$ wants $z = 1$
- $\overline{y} \vee \overline{z}$ wants $z = 0$
- Merge clauses & remove z — must satisfy $x \vee \overline{y}$

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

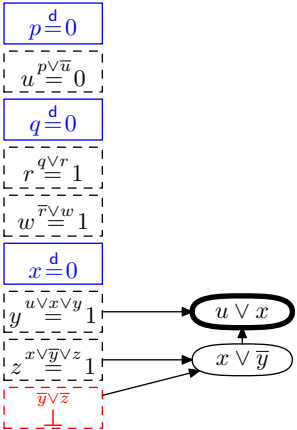
$$(p \vee \overline{u}) \wedge (q \vee r) \wedge (\overline{r} \vee w) \wedge (u \vee x \vee y) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{p} \vee \overline{u})$$



Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

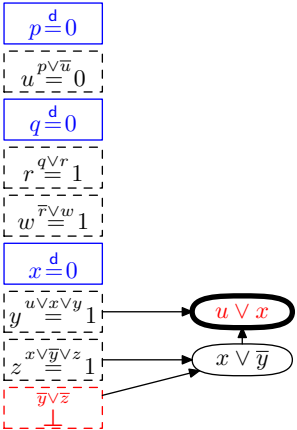


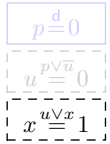
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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





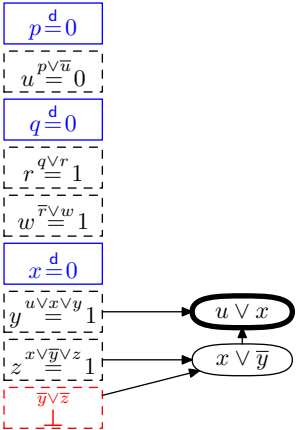
Assertion level 1 (2nd largest level in learned clause) — trim trail to that level

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

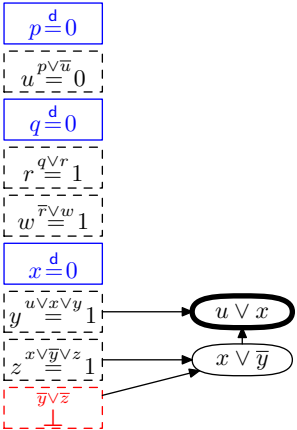
Now UIP literal guaranteed to flip (assert) — but this is a propagation, not a decision

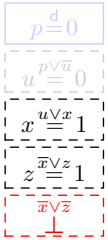
Then continue as before...

Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

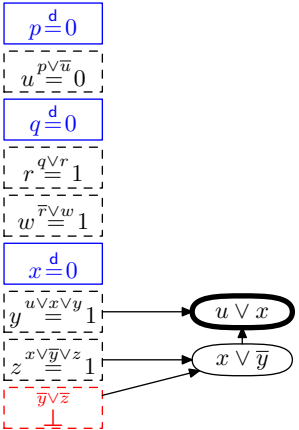


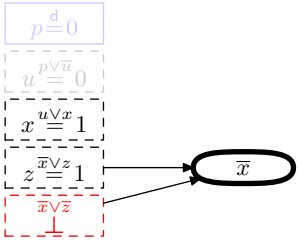


Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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

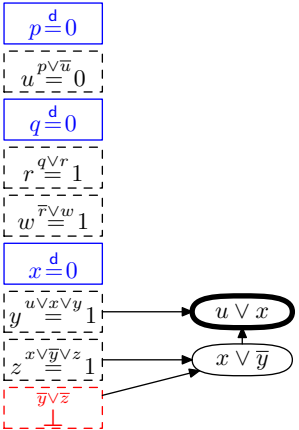


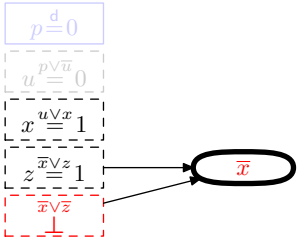


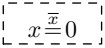
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



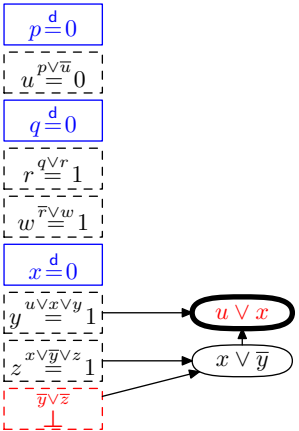


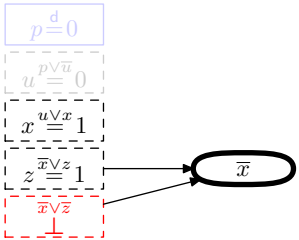


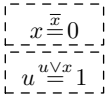
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



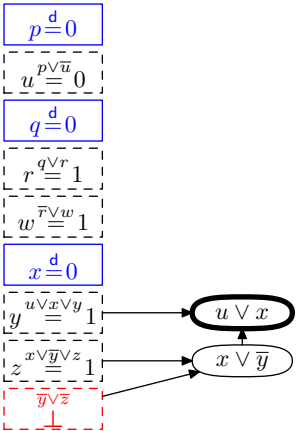


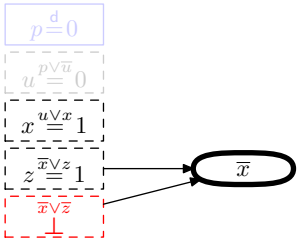


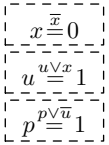
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



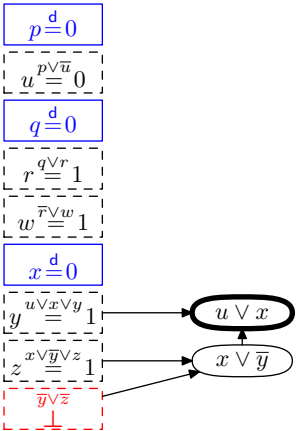


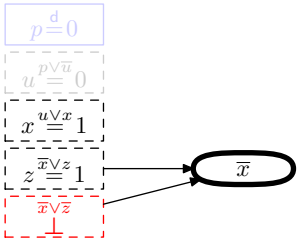


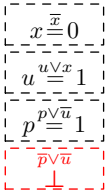
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



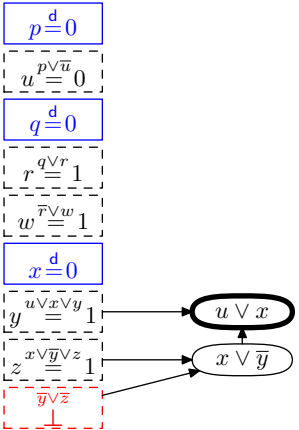


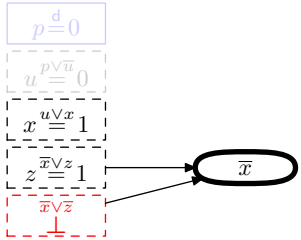


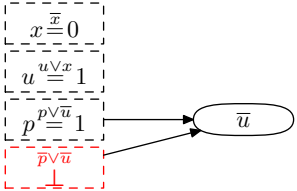
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



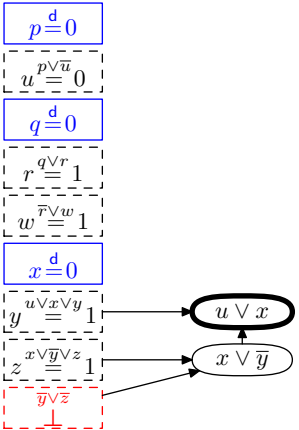


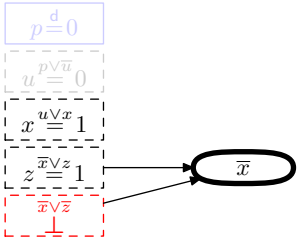


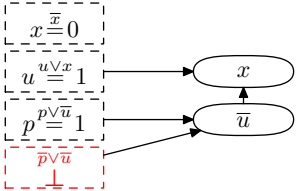
Complete Example of CDCL Execution

Backjump: undo max #decisions while learned clause propagates

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



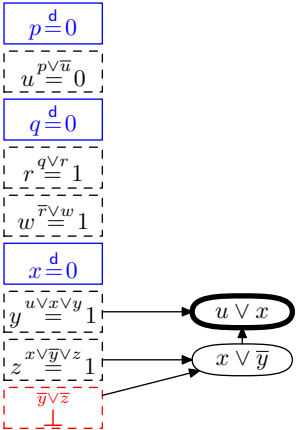


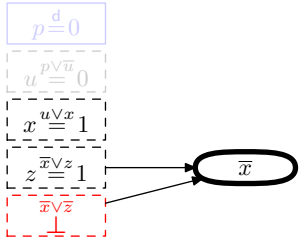


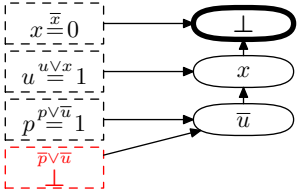
Complete Example of CDCL Execution

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

- In addition to learning clauses, also erase learned clauses that don't seem useful
- Modern solvers do this very aggressively
- Speeds up CDCL search (in particular, unit propagation, which dominates running time)
- But erasing too aggressively can throw away clauses that would have made solver terminate faster [EGG+18]
- So potential trade-off between search speed and search quality
- Except sometimes getting rid of clauses improves search quality too! [KN20]

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
- Restart at fixed intervals or (better) make adaptive restarts depending on "quality" of learned clauses [AS09, AS12]

Conflict-Driven Clause Learning in Pseudocode (Slightly Simplified)

CDCL(F)

$\mathcal{D} \leftarrow F$; // initialize clause database to contain formula
$\rho \leftarrow \emptyset$; // initialize assignment trail to empty
forever do
  if ρ *falsifies some clause* $C \in \mathcal{D}$ **then**
    $A \leftarrow \text{analyzeConflict}(\mathcal{D}, \rho, C)$;
    if $A = \bot$ **then** output UNSATISFIABLE and exit ;
    else add learned clause A to $\mathcal{D}$ and backjump by shrinking ρ ;
  else if *exists clause* $C \in \mathcal{D}$ *unit propagating* x *to* $b \in \{0, 1\}$ *under* ρ **then**
    add propagated assignment $x \overset{C}{=} b$ to ρ ;
  else if *time to restart* **then** $\rho \leftarrow \emptyset$;
  else if *time for clause database reduction* **then**
    erase (roughly) half of learned clauses in $\mathcal{D} \setminus F$ from $\mathcal{D}$
  else if *all variables assigned* **then** output SATISFIABLE and exit ;
  else
    use decision scheme to choose x and b and add assignment $x \overset{\text{d}}{=} b$ to ρ ;

Conflict-Driven Clause Learning in Pseudocode (Slightly Simplified)

CDCL(F)

1 $\mathcal{D} \leftarrow F$; // initialize clause database to contain formula
2 $\rho \leftarrow \emptyset$; // initialize assignment trail to empty
3 **forever do**
4 **if** ρ *falsifies some clause* $C \in \mathcal{D}$ **then**
5 $A \leftarrow$ analyzeConflict$(\mathcal{D}, \rho, C)$;
6 **if** $A = \bot$ **then** output UNSATISFIABLE and exit ;
7 **else** add learned clause A to $\mathcal{D}$ and backjump by shrinking ρ ;
8 **else if** *exists clause* $C \in \mathcal{D}$ *unit propagating* x *to* $b \in \{0, 1\}$ *under* ρ **then**
9 add propagated assignment $x \overset{C}{=} b$ to ρ ;
10 **else if** *time to restart* **then** $\rho \leftarrow \emptyset$;
11 **else if** *time for clause database reduction* **then**
12 erase (roughly) half of learned clauses in $\mathcal{D} \setminus F$ from $\mathcal{D}$
13 **else if** *all variables assigned* **then** output SATISFIABLE and exit ;
14 **else**
15 use decision scheme to choose x and b and add assignment $x \overset{\text{d}}{=} b$ to ρ ;

Conflict Analysis Pseudocode

analyzeConflict$(\mathcal{D}, \rho, C_{\text{confl}})$

```
1 C_learn ← C_confl ;
2 while C_learn not UIP clause and C_learn ≠ ⊥ do
3 |   ℓ ← literal assigned last on trail ρ ;
4 |   if ℓ propagated and ¬ℓ occurs in C_learn then
5 |   |   C_reason ← reason(ℓ, ρ, D) ;
6 |   |   C_learn  ← resolve(C_learn, C_reason) ;
7 |   ρ ← ρ \ {ℓ} ;
8 return C_learn ;
```

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Conflict analysis
- Restarts
- Clause database reduction
- Preprocessing

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Conflict analysis
- Restarts
- Clause database reduction
- Preprocessing

Some natural questions:

- How best to combine these ingredients into a recipe?
- When and why does this recipe work?

State-of-the-art SAT solvers: What About the Recipe?

List of ingredients again (not exhaustive):

- Variable decisions & propagations
- Conflict analysis
- Restarts
- Clause database reduction
- Preprocessing

Some natural questions:

- How best to combine these ingredients into a recipe?
- When and why does this recipe work?

Why SAT solvers actually work so well is poorly understood

Plenty of research has been done to comprehend this better
(Among other places in the MIAO group)

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

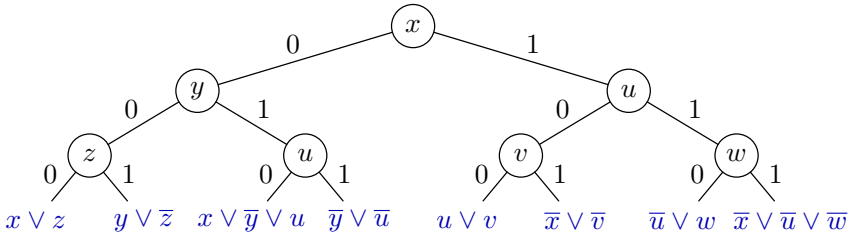
DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

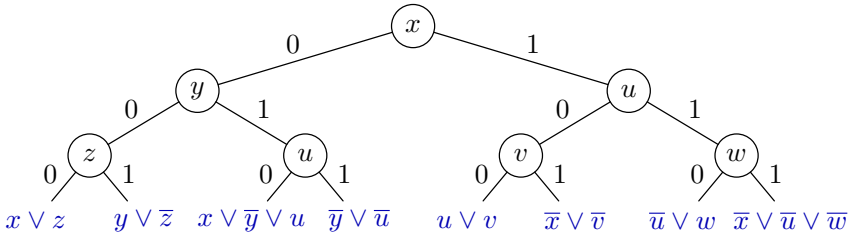
Look at our example again



DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

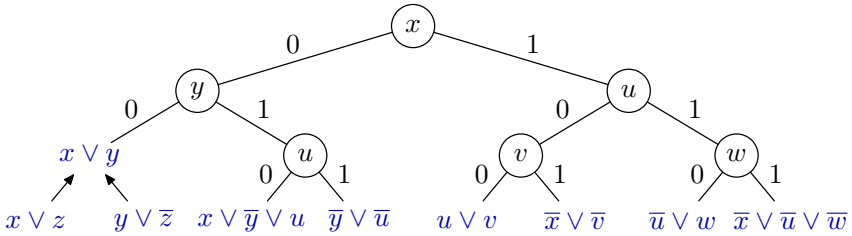


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

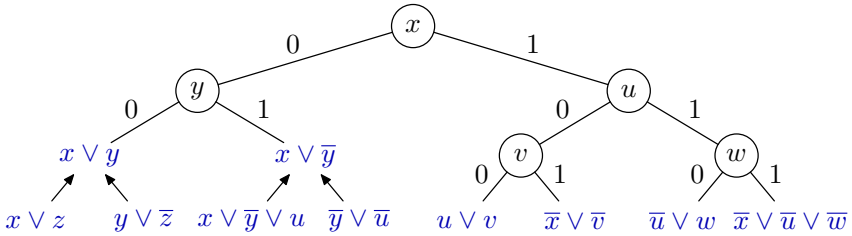


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

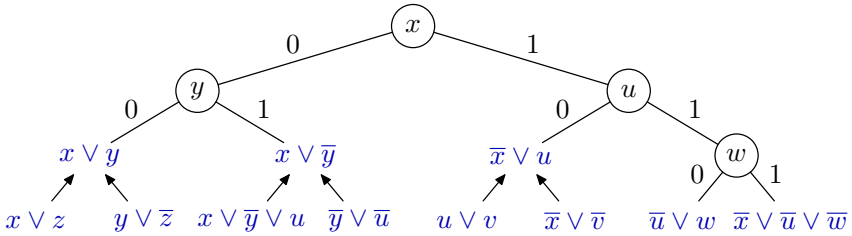


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

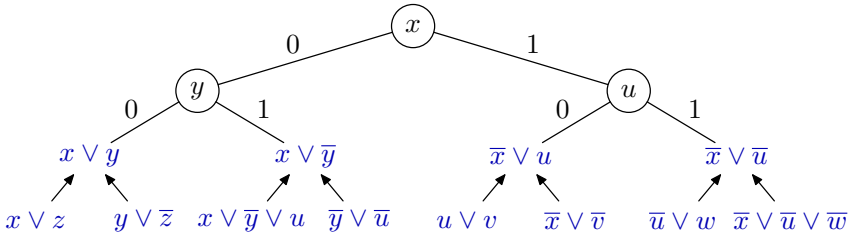


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

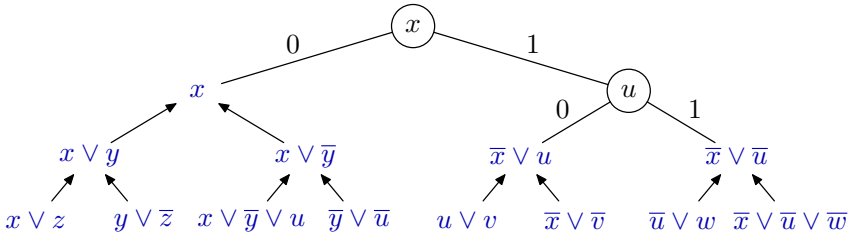


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

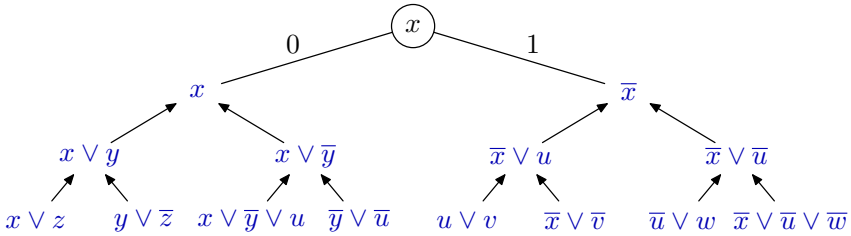


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again

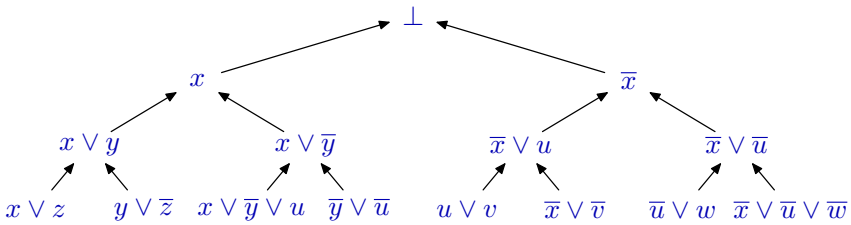


and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL and Resolution Proofs

A DPLL execution is essentially a resolution proof

Look at our example again



and apply resolution rule $\frac{C_1 \vee x \quad C_2 \vee \overline{x}}{C_1 \vee C_2}$ bottom-up

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once
  (to use a clause twice, we have to derive it twice from scratch)

DPLL Running Time and Tree-Like Resolution Proof Size

- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once
  (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$
  lower bounds on DPLL running time

DPLL Running Time and Tree-Like Resolution Proof Size

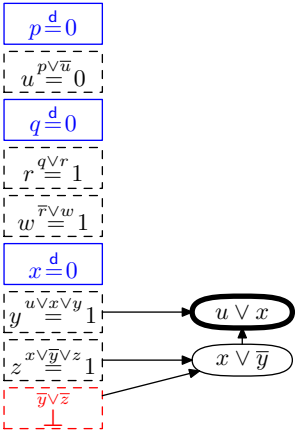
- Can extract resolution proof from any DPLL execution
- Requires an argument, of course, but not too hard to show
- Such proof is tree-like — every derived clause used only once
  (to use a clause twice, we have to derive it twice from scratch)
- Hence, lower bounds on tree-like proof size in resolution $\Rightarrow$
  lower bounds on DPLL running time
- Conflict-driven clause learning adds "shortcut edges" in tree, but still yields resolution proof

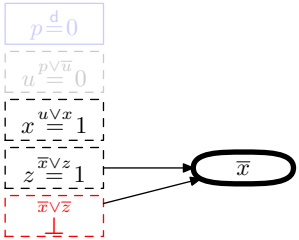
CDCL and Resolution Proofs

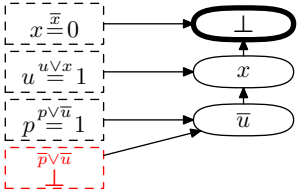
Obtain resolution proof...

CDCL and Resolution Proofs

Obtain resolution proof from our example CDCL execution...

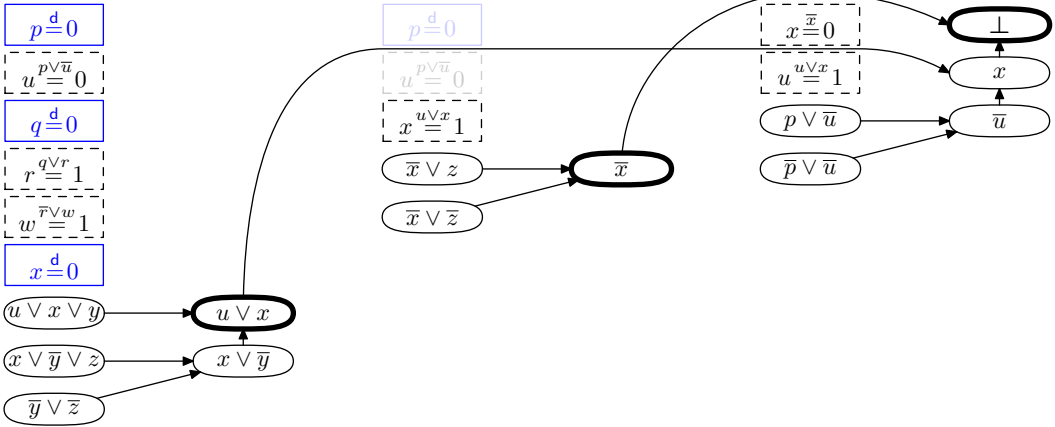






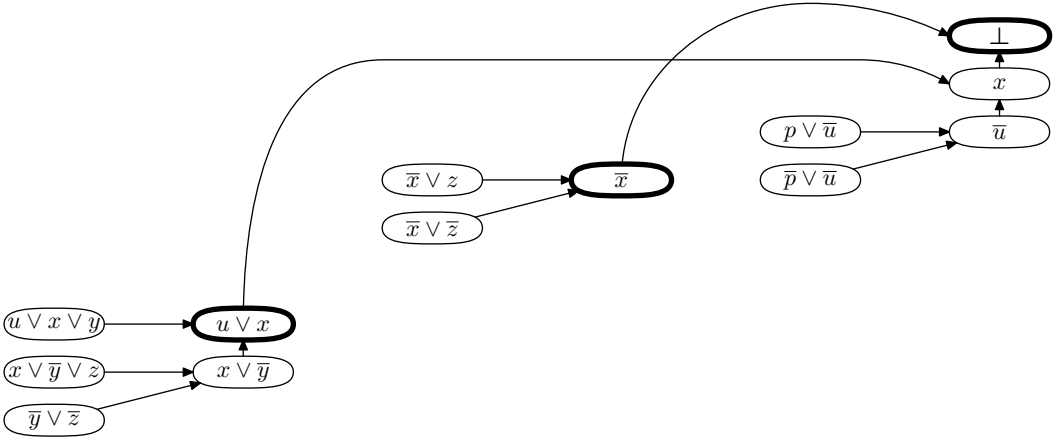
CDCL and Resolution Proofs

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 (DAG-like, not tree-like) from CDCL execution

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required details...

CDCL Running Time and General Resolution Proof Size

- Can extract general resolution proof (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required 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 (DAG-like, not tree-like) from CDCL execution
- Again requires an argument, but you have seen enough in this lecture to be able to fill in the required 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...

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why do heuristics work?
  - Why are applied instances easy?

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why 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, MN14])

Current State of Affairs in SAT Solving

- State-of-the-art CDCL solvers often perform amazingly well ("SAT is easy in practice")
- Very poor theoretical understanding:
  - Why 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, MN14])
- See Chapter 7 on *Proof Complexity and SAT Solving* in the *Handbook of Satisfiability* [BN21] or the lecture `youtu.be/9NR_RGIs1no` for more details

Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{Z}_{\geq 0}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n+1, \ldots, N$

Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{Z}_{\geq 0}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n+1, \ldots, N$

- Linear constraints
- Integer-valued variables
- Real-valued variables
- Linear objective function

Mixed Integer Linear Programming

Mixed integer linear program

- Minimize $\sum_j a_j x_j$
- Subject to $\sum_j a_{i,j} x_j \leq A_i$, $i = 1, \ldots, m$
- $x_j \in \mathbb{Z}_{\geq 0}$ for $j = 1, \ldots, n$
- $x_j \in \mathbb{R}_{\geq 0}$ for $j = n+1, \ldots, N$

- Linear constraints
- Integer-valued variables
- Real-valued variables
- Linear objective function

- No real-valued variables: integer linear program (ILP)
- $0 \leq x_j \leq 1$ for all j: 0-1 ILP
- Vacuous objective $\sum_j 0 \cdot x_j$: decision problem
- But MIP makes most sense for optimization

Constraint Programming

- Variables can also range over enumerations, strings, graphs, sets, ...
- More complex constraints (can be global and NP-hard to propagate)

Constraint Programming

- Variables can also range over enumerations, strings, graphs, sets, ...
- More complex constraints (can be global and NP-hard to propagate)

Example problem:

$$X, Y, Z \in \{1, 2, 3, 4\}$$
$$2Y \neq X + Z$$
$$2X + 3Y + 4Z \leq 16$$
$$\mathit{alldiff}\,(X, Y, Z)$$

Comparing SAT, CP, and MIP

Academia vs. industry

- Best SAT and CP solvers found in academia and/or are open-source — e.g., SAT solvers KISSAT [Kis] and CADICAL [CaD] and CP solver OR-TOOLS [ORT]
- Best MIP solvers are commercial and closed-source — e.g., CPLEX [CPL] (historically), GUROBI [Gur], and FICO XPRESS [FIC]
- Academic MIP solvers SCIP [SCI] and HIGHS [HiG] are excellent but not as good

Comparing SAT, CP, and MIP

Academia vs. industry

- Best SAT and CP solvers found in academia and/or are open-source — e.g., SAT solvers KISSAT [Kis] and CADICAL [CaD] and CP solver OR-TOOLS [ORT]
- Best MIP solvers are commercial and closed-source — e.g., CPLEX [CPL] (historically), GUROBI [Gur], and FICO XPRESS [FIC]
- Academic MIP solvers SCIP [SCI] and HIGHS [HiG] are excellent but not as good

Search vs. backtracking

- SAT: Fast decisions; careful, slow(er) conflict analysis
- MIP: Lots of time & effort on decisions; backtracking not so advanced
- CP: Very powerful propagation algorithms; backtracking again not so advanced

Comparing SAT, CP, and MIP

Academia vs. industry

- Best SAT and CP solvers found in academia and/or are open-source — e.g., SAT solvers KISSAT [Kis] and CADICAL [CaD] and CP solver OR-TOOLS [ORT]
- Best MIP solvers are commercial and closed-source — e.g., CPLEX [CPL] (historically), GUROBI [Gur], and FICO XPRESS [FIC]
- Academic MIP solvers SCIP [SCI] and HIGHS [HiG] are excellent but not as good

Search vs. backtracking

- SAT: Fast decisions; careful, slow(er) conflict analysis
- MIP: Lots of time & effort on decisions; backtracking not so advanced
- CP: Very powerful propagation algorithms; backtracking again not so advanced

So maybe MIP and CP solvers could be improved by adding conflict analysis?

Comparing SAT, CP, and MIP

Academia vs. industry

- Best SAT and CP solvers found in academia and/or are open-source — e.g., SAT solvers KISSAT [Kis] and CADICAL [CaD] and CP solver OR-TOOLS [ORT]
- Best MIP solvers are commercial and closed-source — e.g., CPLEX [CPL] (historically), GUROBI [Gur], and FICO XPRESS [FIC]
- Academic MIP solvers SCIP [SCI] and HIGHS [HiG] are excellent but not as good

Search vs. backtracking

- SAT: Fast decisions; careful, slow(er) conflict analysis
- MIP: Lots of time & effort on decisions; backtracking not so advanced
- CP: Very powerful propagation algorithms; backtracking again not so advanced

So maybe MIP and CP solvers could be improved by adding conflict analysis?
Has been tried, but with much less success than in SAT solving — why?

Approaches for Generalizing Conflict Analysis

Standard approach: Implication graph perspective

- Clauses encode implications between literals
- Learned clauses are cuts in the implication graph
- Extract implications/clauses from more expressive reason constraints
- Perform SAT conflict analysis on the extracted clauses

Approaches for Generalizing Conflict Analysis

Standard approach: Implication graph perspective

- Clauses encode implications between literals
- Learned clauses are cuts in the implication graph
- Extract implications/clauses from more expressive reason constraints
- Perform SAT conflict analysis on the extracted clauses

New approach: Proof system perspective

- Clauses are syntactic constraints
- Learned clauses are inferred by proof system operating on constraints
- Decide on more expressive syntactic representation of reason constraints
- Perform conflict analysis in proof system designed for such syntactic representation

Approaches for Generalizing Conflict Analysis

Standard approach: Implication graph perspective

- Clauses encode implications between literals
- Learned clauses are cuts in the implication graph
- Extract implications/clauses from more expressive reason constraints
- Perform SAT conflict analysis on the extracted clauses

New approach: Proof system perspective

- Clauses are syntactic constraints
- Learned clauses are inferred by proof system operating on constraints
- Decide on more expressive syntactic representation of reason constraints
- Perform conflict analysis in proof system designed for such syntactic representation

From now on focus on MIP and restricted case of 0–1 ILP

The Cutting Planes Proof System

Cutting planes [CCT87] operates on pseudo-Boolean constraints (a.k.a. 0–1 integer linear inequalities) $\sum_i a_i \ell_i \geq A$ with

- $a_i, A \in \mathbb{N}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

The Cutting Planes Proof System

Cutting planes [CCT87] operates on pseudo-Boolean constraints (a.k.a. 0–1 integer linear inequalities) $\sum_i a_i \ell_i \geq A$ with

- $a_i, A \in \mathbb{N}$
- literals ℓ_i: x_i or $\overline{x}_i$ (where $x_i + \overline{x}_i = 1$)

and has derivation rules

Literal axioms $\dfrac{}{\ell_i \geq 0}$

Linear combination $\dfrac{\sum_i a_i \ell_i \geq A \qquad \sum_i b_i \ell_i \geq B}{\sum_i (c_A a_i + c_B b_i)\ell_i \geq c_A A + c_B B} \quad [c_A, c_B > 0]$

Division $\dfrac{\sum_i a_i \ell_i \geq A}{\sum_i \lceil a_i/c \rceil \ell_i \geq \lceil A/c \rceil} \quad [c > 0]$

MIP View of SAT Conflict Analysis

Search

1. Iteratively propagate all assignments implied by reason clauses
2. If no violated clause, decide to assign variable to 0 or 1 and go to step 1
3. Otherwise some clause C violated $\Rightarrow$ trigger **conflict analysis**

MIP View of SAT Conflict Analysis

Search

1. Iteratively propagate all assignments implied by reason clauses
2. If no violated clause, decide to assign variable to 0 or 1 and go to step 1
3. Otherwise some clause C violated $\Rightarrow$ trigger **conflict analysis**

Conflict analysis

- Clauses propagate variables tightly to $\{0, 1\}$ (over the reals), so decisions + reason clauses form infeasible LP

MIP View of SAT Conflict Analysis

Search

1. Iteratively propagate all assignments implied by reason clauses
2. If no violated clause, decide to assign variable to 0 or 1 and go to step 1
3. Otherwise some clause C violated $\Rightarrow$ trigger **conflict analysis**

Conflict analysis

- Clauses propagate variables tightly to $\{0, 1\}$ (over the reals), so decisions + reason clauses form infeasible LP
- Conflict analysis = positive linear combination of reason clauses to cancel propagating literals

MIP View of SAT Conflict Analysis

Search

1. Iteratively propagate all assignments implied by reason clauses
2. If no violated clause, decide to assign variable to 0 or 1 and go to step 1
3. Otherwise some clause C violated $\Rightarrow$ trigger **conflict analysis**

Conflict analysis

- Clauses propagate variables tightly to $\{0, 1\}$ (over the reals), so decisions + reason clauses form infeasible LP
- Conflict analysis = positive linear combination of reason clauses to cancel propagating literals
- Learned clause = Farkas certificate that
  1. has to be clausal
  2. should propagate literal to opposite value when backtracking and restarting **search**

MIP View of SAT Conflict Analysis

Search

1. Iteratively propagate all assignments implied by reason clauses
2. If no violated clause, decide to assign variable to 0 or 1 and go to step 1
3. Otherwise some clause C violated $\Rightarrow$ trigger **conflict analysis**

Conflict analysis

- Clauses propagate variables tightly to $\{0, 1\}$ (over the reals), so decisions + reason clauses form infeasible LP
- Conflict analysis = positive linear combination of reason clauses to cancel propagating literals
- Learned clause = Farkas certificate that
  1. has to be clausal
  2. should propagate literal to opposite value when backtracking and restarting **search**
- Apply coefficient tightening along the way just to keep constraints nicer

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\}$ $\Rightarrow$ Conflict with C_2
(Note: same constraint can propagate several times!)

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2
(Note: same constraint can propagate several times!)

- Add C_1 and C_2 to cancel x_3 and get

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{x_4 \geq 1}$$

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2
(Note: same constraint can propagate several times!)

- Add C_1 and C_2 to cancel x_3 and get

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{x_4 \geq 1}$$

- But $x_4 \geq 1$ is satisfiable — does **not** explain why solver reached a conflict!

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2
(Note: same constraint can propagate several times!)

- Add C_1 and C_2 to cancel x_3 and get

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{x_4 \geq 1}$$

- But $x_4 \geq 1$ is satisfiable — does **not** explain why solver reached a conflict!
- Problem: Constraints don't propagate tightly, so LP is not infeasible

Cancelling Linear Combination of 0–1 Inequalities Doesn't Work!

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2
(Note: same constraint can propagate several times!)

- Add C_1 and C_2 to cancel x_3 and get

$$\frac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{x_4 \geq 1}$$

- But $x_4 \geq 1$ is satisfiable — does **not** explain why solver reached a conflict!
- Problem: Constraints don't propagate tightly, so LP is not infeasible
- Need to tighten propagation to remove solution $\{x_1 = 0, x_2 = x_4 = 1, x_3 = \frac{1}{2}\}$

Tightening Propagation with Division / Chvátal-Gomory Cuts

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2

Tightening Propagation with Division / Chvátal-Gomory Cuts

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\mathsf{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2

1. Add $\overline{\ell}$ to cancel non-falsified literal(s) ℓ not divisible by propagating literal coefficient
2. Divide weakened constraint by propagating literal coefficient
3. Add to conflicting constraint so that propagated literal cancels

Tightening Propagation with Division / Chvátal-Gomory Cuts

$$C_1 \doteq 2x_1 + 2x_2 + 2x_3 + x_4 \geq 4$$
$$C_2 \doteq 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3$$

Trail $\rho = \{x_1 \overset{\text{d}}{=} 0, x_2 \overset{C_1}{=} 1, x_3 \overset{C_1}{=} 1\} \Rightarrow$ Conflict with C_2

1. Add $\overline{\ell}$ to cancel non-falsified literal(s) ℓ not divisible by propagating literal coefficient
2. Divide weakened constraint by propagating literal coefficient
3. Add to conflicting constraint so that propagated literal cancels

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{2x_1 + 2x_2 + 2x_3 + x_4 \geq 4 \qquad \overline{x}_4 \geq 0}{2x_1 + 2x_2 + 2x_3 \geq 3}\ \text{add to cancel } x_4}{x_1 + x_2 + x_3 \geq 2}\ \text{divide by 2}}{2x_1 + 2x_2 + 2x_3 \geq 4}\ \text{multiply by 2} \qquad 2\overline{x}_1 + 2\overline{x}_2 + 2\overline{x}_3 \geq 3}{0 \geq 1}\ \text{add to cancel } x_3$$

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0,1\}$-value over the reals*

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0,1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0, 1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed
4. Unless D satisfies termination criterion (assertiveness, i.e., flipping a literal), set $C := D$ and go to step 1

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0,1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed
4. Unless D satisfies termination criterion (assertiveness, i.e., flipping a literal), set $C := D$ and go to step 1
5. Learn assertive D, i.e., add to solver database of constraints

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0,1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed
4. Unless D satisfies termination criterion (assertiveness, i.e., flipping a literal), set $C := D$ and go to step 1
5. Learn assertive D, i.e., add to solver database of constraints
6. Backjump by undoing further assignments in reverse chronological order until D is no longer violated

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0, 1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed
4. Unless D satisfies termination criterion (assertiveness, i.e., flipping a literal), set $C := D$ and go to step 1
5. Learn assertive D, i.e., add to solver database of constraints
6. Backjump by undoing further assignments in reverse chronological order until D is no longer violated
7. Switch back to **search** phase (which starts with D propagating)

MIP View of Pseudo-Boolean Conflict Analysis (Simplified Description)

1. Find reason constraint R responsible for propagating last variable x in violated constraint C to "wrong value"
2. Apply cut rule to generate (globally valid) constraint R_{cut} propagating x to $\{0,1\}$-value over the reals*
3. Set $D :=$ smallest integer linear combination of R_{cut} and C for which x cancels — then D must be violated by current trail with x removed
4. Unless D satisfies termination criterion (assertiveness, i.e., flipping a literal), set $C := D$ and go to step 1
5. Learn assertive D, i.e., add to solver database of constraints
6. Backjump by undoing further assignments in reverse chronological order until D is no longer violated
7. Switch back to **search** phase (which starts with D propagating)

(*) Finding such a cut is always possible, but this requires an argument

Comparison of Propagation and Conflict Analysis in SAT and MIP

Propagation in MIP solvers

- Fast, simple propagation as in SAT solvers
- Plus powerful, but slower, method of solving LP relaxations

Comparison of Propagation and Conflict Analysis in SAT and MIP

Propagation in MIP solvers

- Fast, simple propagation as in SAT solvers
- Plus powerful, but slower, method of solving LP relaxations

Conflict analysis in MIP solvers [Ach07, ABKW08, AW13]

- Perform derivation not on reason constraints R as described above
- Instead use disjunctive clauses extracted from reason constraints
- Incurs exponential loss in theoretical reasoning power compared to operating on actual linear constraints (follows from [BKS04, CCT87, Hak85])
- Proof-of-concept implementation shows that new conflict analysis can be better also in practice [MBGN23, MSB+25]

Open Problems for Conflict Analysis in Stronger Paradigms than SAT

Pseudo-Boolean (PB) conflict analysis on 0–1 linear inequalities

- Many more degrees of freedom than SAT conflict analysis
- Many (most?) aspects still unexplored
- Which other MIP cut rules could be useful for tightening propagations?

Open Problems for Conflict Analysis in Stronger Paradigms than SAT

Pseudo-Boolean (PB) conflict analysis on 0–1 linear inequalities

- Many more degrees of freedom than SAT conflict analysis
- Many (most?) aspects still unexplored
- Which other MIP cut rules could be useful for tightening propagations?

General MIP conflict analysis

- PB conflict analysis can be extended to 0–1 mixed linear programs [MSB$^+$25]
- But room for lots of additional tuning
- And dealing with general integer-valued variables seems hard

Open Problems for Conflict Analysis in Stronger Paradigms than SAT

Pseudo-Boolean (PB) conflict analysis on 0–1 linear inequalities

- Many more degrees of freedom than SAT conflict analysis
- Many (most?) aspects still unexplored
- Which other MIP cut rules could be useful for tightening propagations?

General MIP conflict analysis

- PB conflict analysis can be extended to 0–1 mixed linear programs [MSB$^+$25]
- But room for lots of additional tuning
- And dealing with general integer-valued variables seems hard

Conflict analysis for constraint programming

- Lazy clause generation CP solvers [OSC09] suffer from same problem as MIP solvers
- Reasons for propagations can also be generated as 0–1 linear constraints and used to certify correctness of solver reasoning [EGMN20, GMN22, MM23, MMN24, MM25]
- Could such constraints also be used efficiently for pseudo-Boolean conflict analysis?

Research Goals in the MIAO Group (1/2)

Strengthen the mathematical analysis of algorithmic methods

- Study methods of reasoning powerful enough to capture state-of-the-art algorithms used in practice
- Prove theorems about their power and limitations
- E.g., look for analogues of result that resolution proof system captures CDCL reasoning

Research Goals in the MIAO Group (1/2)

Strengthen the mathematical analysis of algorithmic methods

- Study methods of reasoning powerful enough to capture state-of-the-art algorithms used in practice
- Prove theorems about their power and limitations
- E.g., look for analogues of result that resolution proof system captures CDCL reasoning

Construct stronger algorithms for combinatorial problems

- Use insights into stronger mathematical methods of reasoning to build algorithms for SAT and other combinatorial problems
- Aiming for exponential speed-ups over state of the art
- E.g., use cutting-planes-based reasoning to build pseudo-Boolean solvers

Research Goals in the MIAO Group (2/2)

Improve understanding of efficient computation in practice

- Use computational complexity theory to study "real-world" (not worst-case) problems
- Combine theoretical study and empirical experiments
- E.g., take "crafted formulas" with provable theoretical properties and investigate correlation with practical solver performance

Research Goals in the MIAO Group (2/2)

Improve understanding of efficient computation in practice

- Use computational complexity theory to study "real-world" (not worst-case) problems
- Combine theoretical study and empirical experiments
- E.g., take "crafted formulas" with provable theoretical properties and investigate correlation with practical solver performance

Certify correctness for modern combinatorial solvers

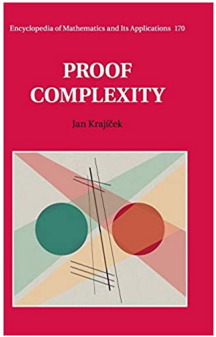
- In many combinatorial optimization paradigms, state-of-the-art solvers are known to be buggy (definitely holds for MIP and CP)
- Develop methods to make solvers output not just answer but machine-verifiable proof of correctness of this answer
- Work on, e.g., SAT-based optimization (MaxSAT), pseudo-Boolean solving, subgraph solving, constraint programming, and automated planning

Some References for In-Depth Reading

Handbook of Satisfiability
(Especially chapter 7 ☺)

[BHvMW21]

Proof Complexity
by Jan Krajíček



[Kra19]

And survey papers, slides, and videos at `www.jakobnordstrom.se`

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Designing conflict analysis methods for CP and MIP is a very interesting challenge!

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Designing conflict analysis methods for CP and MIP is a very interesting challenge!
- Lots of exciting work for PhD students and postdocs
  (we're hiring! ☺)

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously efficient in practice
- SAT solving more of an art form than a science — theoretical understanding lagging far behind
- Can use proof complexity to analyze potential and limitations of SAT solvers
- And to get inspirations for algorithms based on stronger methods of reasoning
- Designing conflict analysis methods for CP and MIP is a very interesting challenge!
- Lots of exciting work for PhD students and postdocs
  (we're hiring! ☺)

Thanks for listening!

References I

[AAB+14] David Allouche, Isabelle André, Sophie Barbe, Jessica Davies, Simon de Givry, George Katsirelos, Barry O'Sullivan, Steve Prestwich, Thomas Schiex, and Seydou Traoré. Computational protein design as an optimization problem. *Artificial Intelligence*, 212(1):59–79, July 2014.

[ABFP12] Cyril Allignol, Nicolas Barnier, Pierre Flener, and Justin Pearson. Constraint programming for air traffic management: A survey. *The Knowledge Engineering Review*, 27(3):361–392, July 2012.

[ABKW08] Tobias Achterberg, Timo Berthold, Thorsten Koch, and Kati Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In *Proceedings of the 5th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR '08)*, volume 5015 of *Lecture Notes in Computer Science*, pages 6–20. Springer, May 2008.

[Ach07] Tobias Achterberg. Conflict analysis in mixed integer programming. *Discrete Optimization*, 4(1):4–20, March 2007.

[AGH+22] Elvira Albert, Pablo Gordillo, Alejandro Hernández-Cerezo, Clara Rodríguez-Núñez, and Albert Rubio. Using automated reasoning techniques for enhancing the efficiency and security of (Ethereum) smart contracts. In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*, volume 13385 of *Lecture Notes in Computer Science*, pages 3–7. Springer, August 2022.

References II

[AGRS20] Elvira Albert, Pablo Gordillo, Albert Rubio, and Maria A. Schett. Synthesis of super-optimized smart contracts using Max-SMT. In *Proceedings of the 32nd International Conference on Computer Aided Verification (CAV '20)*, volume 12224 of *Lecture Notes in Computer Science*, pages 177–200. Springer, July 2020.

[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.

[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.

[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.

References III

[BKS04] Paul Beame, Henry Kautz, and Ashish Sabharwal. Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351, December 2004. Preliminary version in *IJCAI '03*.

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

[BN21] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [BHvMW21], chapter 7, pages 233–350.

[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.

[BvdKM+21] Péter Biró, Joris van de Klundert, David F. Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits C. R. Spieksma, María O. Valentín, and Ana Viana. Modelling and optimisation in European kidney exchange programmes. *European Journal of Operational Research*, 291(2):447–456, June 2021.

References IV

[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*.

[CaD] CaDiCaL simplified satisfiability solver. `https://github.com/arminbiere/cadical`.

[CCT87] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, November 1987.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Revised Selected Papers from the 4th International Workshop on Parameterized and Exact Computation (IWPEC '09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 75–85. Springer, September 2009.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC '71)*, pages 151–158, May 1971.

[CPL] IBM ILOG CPLEX optimization studio. `https://www.ibm.com/products/ilog-cplex-optimization-studio`.

References V

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

[DMB11] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, 54(9):69–77, September 2011.

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

[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.

[EGMN20] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, February 2020.

[FFH+16] Andreas Falkner, Gerhard Friedrich, Alois Haselböck, Gottfried Schenner, and Herwig Schreiner. Twenty-five years of successful application of constraint technologies at Siemens. *AI Magazine*, 37(4):67–80, 2016.

References VI

[FIC] FICO Xpress optimization. `https://www.fico.com/en/products/fico-xpress-optimization`.

[GMN22] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. An auditable constraint programming solver. In *Proceedings of the 28th International Conference on Principles and Practice of Constraint Programming (CP '22)*, volume 235 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 25:1–25:18, August 2022.

[Gur] Gurobi optimizer. `https://www.gurobi.com/`.

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

[HD19] Mark A. Hallen and Bruce Randall Donald. Protein design by provable algorithms. *Communications of the ACM*, 62(10):76–84, October 2019.

[HiG] HiGHS – high performance software for linear optimization. `https://highs.dev/`.

[HK17] Marijn J. H. Heule and Oliver Kullmann. The science of brute force. *Communications of the ACM*, 60(8):70–79, August 2017.

References VII

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, March 2001. Preliminary version in *CCC '99*.

[Kis] The Kissat SAT solver. `https://github.com/arminbiere/kissat`.

[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.

[Kra19] Jan Krajíček. *Proof Complexity*, volume 170 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, March 2019.

[Lev73] Leonid A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973. In Russian. Available at `http://mi.mathnet.ru/ppi914`.

[Man16] David F. Manlove. Hospitals/residents problem. In Ming-Yang Kao, editor, *Encyclopedia of Algorithms*, pages 926–930. Springer New York, 2016.

References VIII

[MBGN23] Gioni Mexi, Timo Berthold, Ambros Gleixner, and Jakob Nordström. Improving conflict analysis in MIP solvers by pseudo-Boolean reasoning. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–26:19, August 2023.

[MM23] Matthew McIlree and Ciaran McCreesh. Proof logging for smart extensional constraints. In *Proceedings of the 29th International Conference on Principles and Practice of Constraint Programming (CP '23)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 26:1–26:17, August 2023.

[MM25] Matthew McIlree and Ciaran McCreesh. Certifying bounds propagation for integer multiplication constraints. In *Proceedings of the 39th AAAI Conference on Artificial Intelligence (AAAI '25)*, pages 11309–11317, February-March 2025.

[MMN24] Matthew McIlree, Ciaran McCreesh, and Jakob Nordström. Proof logging for the circuit constraint. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14743 of *Lecture Notes in Computer Science*, pages 38–55. Springer, May 2024.

References IX

[MMT17] David F. Manlove, Iain McBride, and James Trimble. "Almost-stable" matchings in the hospitals / residents problem with couples. *Constraints*, 22(1):50–72, January 2017.

[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.

[MN14] Mladen Mikša and Jakob Nordström. Long proofs of (seemingly) simple formulas. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 121–137. Springer, July 2014.

[MO12] David F. Manlove and Gregg O'Malley. Paired and altruistic kidney donation in the UK: Algorithms and experimentation. In *Proceedings of the 11th International Symposium on Experimental Algorithms (SEA '12)*, volume 7276 of *Lecture Notes in Computer Science*, pages 271–282. Springer, June 2012.

[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*.

References X

[MSB+25] Gioni Mexi, Felipe Serrano, Timo Berthold, Ambros Gleixner, and Jakob Nordström. Cut-based conflict analysis in mixed integer programming. *INFORMS Journal on Computing*, 2025. To appear.

[ORT] Google OR-Tools. `https://developers.google.com/optimization`.

[OSC09] Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, January 2009. Preliminary versions of these results appeared in *CP '07* and *CATS '08*.

[PBD+22] Lise Pomiès, Céline Brouard, Harold Duruflé, Élise Maigné, Clément Carré, Louise Gody, Fulya Trösser, George Katsirelos, Brigitte Mangin, Nicolas B. Langlade, and Simon de Givry. Gene regulatory network inference methodology for genomic and transcriptomic data acquired in genetically related heterozygote individuals. *Bioinformatics*, 38(17):4127–4134, September 2022.

[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.

References XI

[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.

[SCI] SCIP: Solving constraint integer programs. `https://scipopt.org`.

[Sha09] Natarajan Shankar. Automated deduction for verification. *ACM Computing Surveys*, 41(4):20:1–20:56, October 2009.

[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.

[ZR14] Yang Zhao and Kristin Yvonne Rozier. Formal specification and verification of a coordination protocol for an automated air traffic control system. *Science of Computer Programming*, 96:337–353, December 2014.