Solving Exponentially Hard Problems in Linear Time(?)

Jakob Nordström

University of Copenhagen

DIKU Bits
December 3, 2019

This Is Me. . .

Jakob Nordström

Associate Professor

Algorithms and Complexity Section

Department of Computer Science (DIKU)

University of Copenhagen

jn@di.ku.dk

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

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

Three Simple Problems...

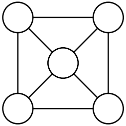
Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

Three Simple Problems. . .

Colouring

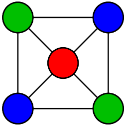
Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?



Three Simple Problems. . .

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

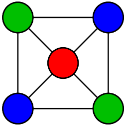


3-colouring exists

Three Simple Problems. . .

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?



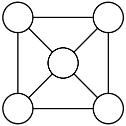
3-colouring exists but no 2-colouring

Three Simple Problems. . .

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

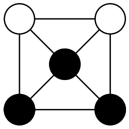
Three Simple Problems. . .



Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems. . .

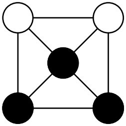


3-clique exists

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems. . .



3-clique exists but no 4-clique

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Three Simple Problems...

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Sat

Given propositional logic formula, is there a satisfying assignment?

Three Simple Problems...

Colouring

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

Clique

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

Sat

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u})$$
$$\land\ (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Three Simple Problems...

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

$$(x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u})$$
$$\land\ (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

- Variables should be set to **true** or **false**
- Constraint $(x \lor \overline{y} \lor z)$: means x or z should be true or y false
- $\land$ means all constraints should hold simultaneously
- Is there a truth value assignment satisfying all constraints?

Three Simple Problems...

COLOURING

Does graph $G = (V, E)$ have a colouring with k colours so that neighbours have distinct colours?

CLIQUE

Is there a clique in graph $G = (V, E)$ with k vertices that are all pairwise connected?

SAT

Given propositional logic formula, is there a satisfying assignment?

COLOURING:	frequency allocation for mobile base stations
CLIQUE:	bioinformatics, computational chemistry
SAT:	easily models these and many other problems

. . . with Huge Practical Implications

- Some examples of problems that can be encoded as logic formulas:
  - computer hardware verification
  - computer software testing
  - artificial intelligence
  - cryptography
  - bioinformatics
  - et cetera. . .
- Leads to **humongous** formulas (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

Solving Logic Formulas in Practice

- Dramatic progress last 15–20 years on so-called SAT solvers
  Today routinely used to solve large-scale real-world problems
- But... There are also small formulas (just ∼100 variables) that are completely beyond reach of even the very best SAT solvers
- Best known algorithms based on fairly simple method from early 1960s (Davis-Putnam-Logemann-Loveland or DPLL)
- How do these SAT solvers work?
- How can they be so good in practice?
- When they fail to be efficient, can we understand why?
- It's 2019 now — can we go beyond techniques from 1960s?

Plan for the Rest of This Talk

What we hope to cover in this talk:

Plan for the Rest of This Talk

What we hope to cover in this talk:

- Define more precisely the computational problem

Plan for the Rest of This Talk

What we hope to cover in this talk:

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

Plan for the Rest of This Talk

What we hope to cover in this talk:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance

Plan for the Rest of This Talk

What we hope to cover in this talk:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance
- Discuss possible ways to go beyond current state of the art

Plan for the Rest of This Talk

What we hope to cover in this talk:

- Define more precisely the computational problem
- Give (simplified) description of how modern SAT solvers work
- Present tools to analyze SAT solver performance
- Discuss possible ways to go beyond current state of the art

. . . And in the process also touch on some of the research being done in the Algorithms and Complexity Section at DIKU

Formal Description of Problem

- Literal a: variable x or its negation $\overline{x}$
- Clause $C = a_1 \vee \cdots \vee a_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- CNF formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses

Formal Description of Problem

- Literal a: variable x or its negation $\overline{x}$
- Clause $C = a_1 \lor \cdots \lor a_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- CNF formula $F = C_1 \land \cdots \land C_m$: conjunction of clauses

The SATISFIABILITY (or just SAT) Problem

Given a CNF formula F, is it satisfiable?

Formal Description of Problem

- Literal a: variable x or its negation $\overline{x}$
- Clause $C = a_1 \vee \cdots \vee a_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)
- 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?

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

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be either true or false, so all in all get 2^n different cases

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be either true or false, so all in all get 2^n different cases
- If formula contains, say, one million variables, we get $2^{1,000,000}$ cases (a number with more than 300,000 digits)

How to Solve the SAT Problem?

- Let computer check all possible assignments! Isn't this exactly the kind of monotone routine work at which computers excel?
- But how many cases to check?
- Suppose formula has n variables
- Each variable can be either true or false, so all in all get 2^n different cases
- If formula contains, say, one million variables, we get $2^{1,000,000}$ cases (a number with more than 300,000 digits)

To understand how large this number is, consider that even if every atom in the known universe was a modern supercomputer that had been running at full speed ever since the beginning of time some 13.7 billion years ago, all of them together would only have covered a completely negligible fraction of these cases by now. So we really would not have time to wait for them to finish...

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time
- So SAT problem might seem hard to solve, but verifying a solution is easy (not all problems have this property — how do you verify a winning position in chess?)

An Interesting Feature of the SAT Problem

- Deciding whether a satisfying assignment exists may take a long time
- But if you happen to know a satisfying assignment, easy to convince someone else that formula is satisfiable
- How? Just give assignment — can be verified in linear time
- So SAT problem might seem hard to solve, but verifying a solution is easy (not all problems have this property — how do you verify a winning position in chess?)
- The family of problems for which solutions are easy to check have a name: NP

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know
- This one of the million dollar "Millennium Problems" posed as the main challenges for mathematics in the new millennium

How to Solve the SAT Problem, Take 2

- The SAT problem can be used to describe any problem in NP
- If you can solve SAT efficiently, then you can solve any problem in NP efficiently (this is why SAT is so useful)
- So how hard is it to solve SAT? (Ok, brute force didn't work, but it usually doesn't — maybe can do something smarter?)
- We don't know
- This one of the million dollar "Millennium Problems" posed as the main challenges for mathematics in the new millennium
- Widely believe to be impossible to solve efficiently on computer in the worst case, but we really don't know

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

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

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

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

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

- If F contains an 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

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

- If F contains an 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

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

- If F contains an 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

An Attempt at a Smarter Case Analysis: DPLL

Ok, but suppose you're out there in reality and actually have to solve the problem — then what do you do?

Chances are you'll use some variant of the DPLL method based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

DPLL (somewhat simplified description)

- If F contains an 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

$$
\begin{aligned}
F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})
\end{aligned}
$$

A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \lor z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

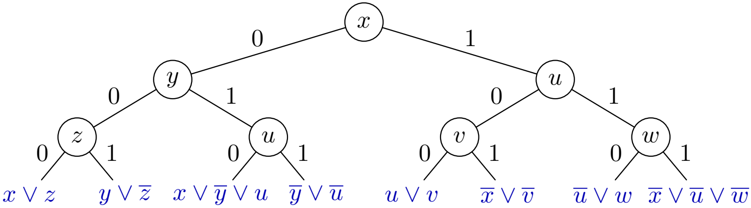
Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

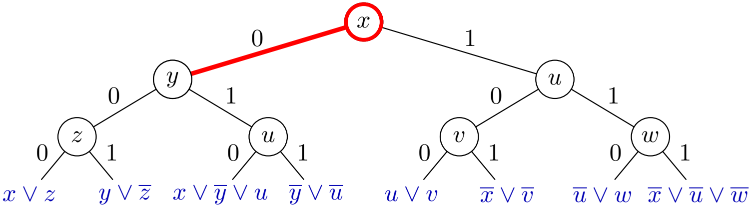


A DPLL Toy Example

$$\begin{aligned} F = \quad & (\qquad z) \land (y \lor \overline{z}) \land (\qquad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\ & \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

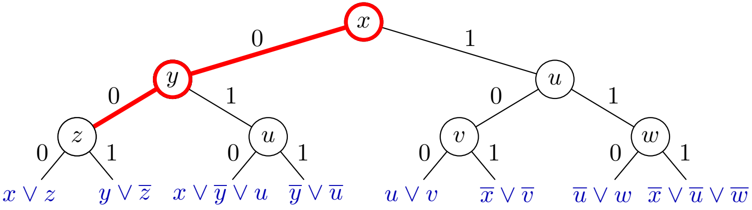


A DPLL Toy Example

$$
\begin{aligned}
F = \quad & (\quad z) \land (\quad \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \\
& \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})
\end{aligned}
$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

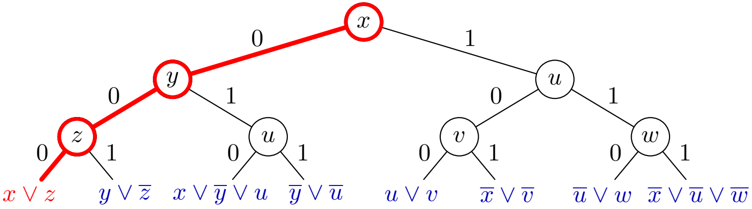


A DPLL Toy Example

$$F = (x \lor z) \land (\overline{z}) \land (\overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

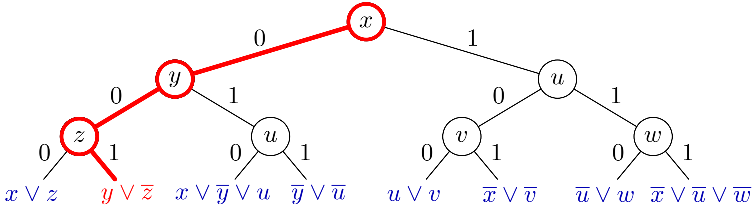


A DPLL Toy Example

$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad \overline{y} \lor u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

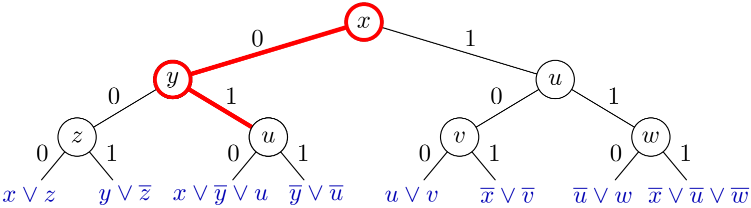


A DPLL Toy Example

$$F = (\quad z) \land (y \lor \overline{z}) \land (\quad u) \land (\quad \overline{u})$$
$$\land\ (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

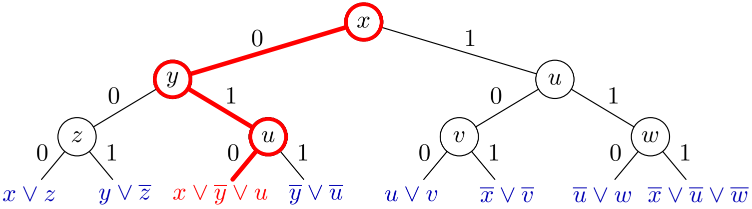


A DPLL Toy Example

$$F = (\quad z) \land (y \lor \overline{z}) \land (x \lor \overline{y} \lor u) \land (\quad \overline{u})$$
$$\land (\quad v) \land (\overline{x} \lor \overline{v}) \land (\overline{u} \lor w) \land (\overline{x} \lor \overline{u} \lor \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

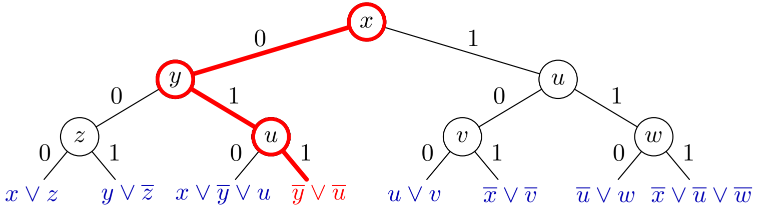


A DPLL Toy Example

$$F = (\qquad z) \land (y \lor \overline{z}) \land (\qquad\qquad u) \land (\overline{y} \lor \overline{u}) \land (u \lor v) \land (\overline{x} \lor \overline{v}) \land (\qquad w) \land (\overline{x} \lor \qquad \overline{w})$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)

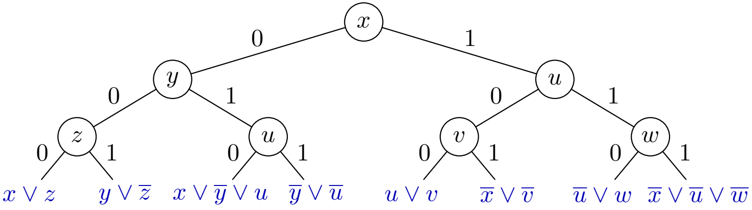


A DPLL Toy Example

$$\begin{aligned} F = \quad & (x \vee z) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{y} \vee u) \wedge (\overline{y} \vee \overline{u}) \\ & \wedge\, (u \vee v) \wedge (\overline{x} \vee \overline{v}) \wedge (\overline{u} \vee w) \wedge (\overline{x} \vee \overline{u} \vee \overline{w}) \end{aligned}$$

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsified clause found (i.e., when conflict reached)



State-of-the-art SAT solvers: Ingredients

Many more ingredients in modern SAT solvers, for instance:

- Branching or decision heuristic (choice of pivot variables crucial)
- When reaching leaf, compute reason for conflict and add to formula as new clause (conflict-driven clause learning (CDCL))
- Every once in a while, restart from beginning (but save computed info)

Let us briefly discuss these ingredients

Variable Decision Heuristics

Unit propagation

- Suppose current assignment falsifies all literals in clause $a_1 \lor a_2 \lor \cdots \lor a_k$ except one (say a_k) — clause is unit
- Then a_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 Decision Heuristics

Unit propagation

- Suppose current assignment falsifies all literals in clause $a_1 \lor a_2 \lor \cdots \lor a_k$ except one (say a_k) — clause is unit
- Then a_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 unit propagations, pick variable with highest score

Conflict-Driven Clause Learning (CDCL)

- At conflict, want to add clause avoiding same part of search tree being explored again
- Suppose decisions $x = 1$, $y = 0$, $z = 1$ led to conflict
- Then can add $\overline{x} \vee y \vee \overline{z}$ to avoid these decisions being made again — decision learning scheme
- In practice, more advanced learning schemes
- Derive new clause from clauses unit propagating on the way to conflict (using resolution, which we will talk about soon)

Restarts

- Every once in a while, 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
- Are even set to same values (phase saving)
- Current intution: improves the search by focusing on important variables
- How often to restart: at fixed intervals or (better) depending on “quality” of learned clauses

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

List of ingredients again (not exhaustive):

- Variable decisions
- Clause learning
- Restarts

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

List of ingredients again (not exhaustive):

- Variable decisions
- Clause learning
- Restarts

Some natural questions:

- How 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
- Clause learning
- Restarts

Some natural questions:

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

Why SAT solvers actually work so well poorly understood question
Lots of research to comprehend this better
(Among other places in the AC Section at DIKU)

Proof Complexity

Proof search algorithm: defines proof system with derivation rules

Proof complexity: study of proofs in such systems

- Lower bounds: no algorithm can do better (even optimal one always guessing the right next step)
- Upper bounds: gives hope for good algorithms if we can search for proofs in system efficiently

Resolution

Resolution rule:

$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

Resolution

Resolution rule:

$$\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$$

Observation

If F is a satisfiable CNF formula and D is derived from clauses $C_1, C_2 \in F$ by the resolution rule, then $F \wedge D$ is satisfiable.

Resolution

Resolution rule:

$$\frac{B \lor x \quad C \lor \overline{x}}{B \lor C}$$

Observation

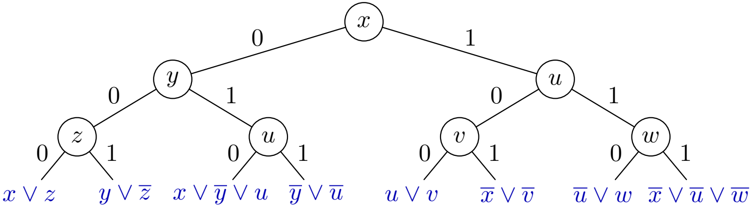
If F is a satisfiable CNF formula and D is derived from clauses $C_1, C_2 \in F$ by the resolution rule, then $F \land D$ is satisfiable.

Prove F unsatisfiable by deriving the unsatisfiable empty clause $\perp$ from F by resolution

DPLL and Resolution

A DPLL execution is essentially a resolution proof

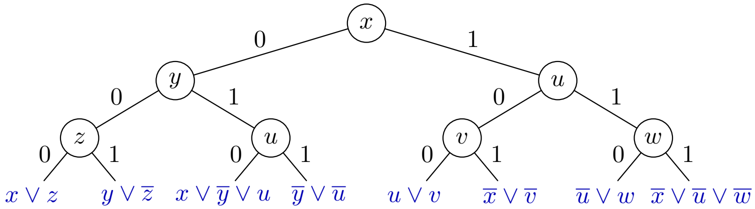
Look at our example again



DPLL and Resolution

A DPLL execution is essentially a resolution proof

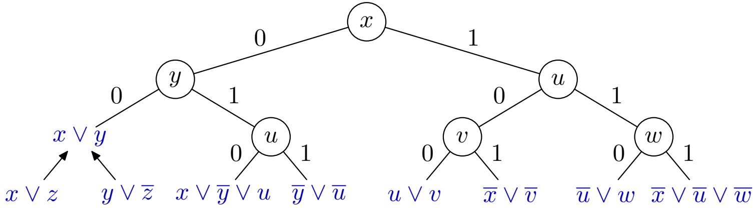
Look at our example again



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

DPLL and Resolution

A DPLL execution is essentially a resolution proof
Look at our example again

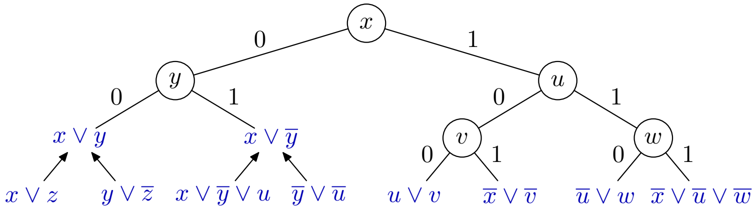


and apply resolution rule $\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$ bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again

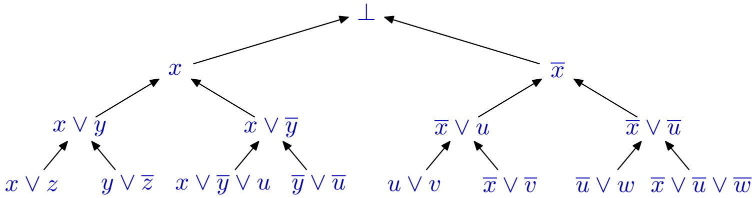


and apply resolution rule $\frac{B \vee x \quad C \vee \overline{x}}{B \vee C}$ bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again



and apply resolution rule $\frac{B \lor x \quad C \lor \overline{x}}{B \lor C}$ bottom-up

Running Time and Proof Size

- Can extract resolution proof from any DPLL execution
- Conflict-driven clause learning adds "shortcut edges" in tree
- But still yields resolution proof
- (Almost) true for other optimizations used by modern SAT solvers as well
- Hence, lower bounds on resolution proof size $\Rightarrow$ lower bounds on SAT solver running time

Examples of Hard Formulas For Resolution (1/3)

Pigeonhole principle (PHP) [Haken '85]
"$n + 1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon $i \rightarrow$ hole j"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

$p_{i,1} \lor p_{i,2} \lor \cdots \lor p_{i,n}$	every pigeon i gets a hole
$\overline{p}_{i,j} \lor \overline{p}_{i',j}$	no hole j gets two pigeons $i \neq i'$

Can also add "functionality" and "onto" axioms

$\overline{p}_{i,j} \lor \overline{p}_{i,j'}$	no pigeon i gets two holes $j \neq j'$
$p_{1,j} \lor p_{2,j} \lor \cdots \lor p_{n+1,j}$	every hole j gets a pigeon

Examples of Hard Formulas For Resolution (1/3)

Pigeonhole principle (PHP) [Haken '85]
"$n+1$ pigeons don't fit into n holes"

Variables $p_{i,j}$ = "pigeon $i \rightarrow$ hole j"; $1 \leq i \leq n+1$; $1 \leq j \leq n$

$p_{i,1} \vee p_{i,2} \vee \cdots \vee p_{i,n}$	every pigeon i gets a hole
$\overline{p}_{i,j} \vee \overline{p}_{i',j}$	no hole j gets two pigeons $i \neq i'$

Can also add "functionality" and "onto" axioms

$\overline{p}_{i,j} \vee \overline{p}_{i,j'}$	no pigeon i gets two holes $j \neq j'$
$p_{1,j} \vee p_{2,j} \vee \cdots \vee p_{n+1,j}$	every hole j gets a pigeon

Even onto functional PHP hard — **"resolution cannot count"**

Resolution proof requires $\exp(\Omega(n)) = \exp\big(\Omega\big(\sqrt[3]{N}\big)\big)$ clauses
(measured in terms of formula size N)

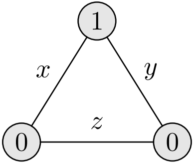
Examples of Hard Formulas For Resolution (2/3)

Tseitin formulas [Urquhart '87]

"Sum of degrees of vertices in graph is even"

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



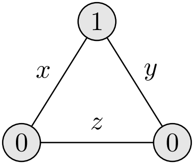
$$
\begin{array}{rr}
(x \lor y) & \land\ (\overline{x} \lor z) \\
\land\ (\overline{x} \lor \overline{y}) & \land\ (y \lor \overline{z}) \\
\land\ (x \lor \overline{z}) & \land\ (\overline{y} \lor z)
\end{array}
$$

Examples of Hard Formulas For Resolution (2/3)

Tseitin formulas [Urquhart '87]
"Sum of degrees of vertices in graph is even"

Variables = edges (in undirected graph of bounded degree)

- Label every vertex 0/1 so that sum of labels odd
- Write CNF requiring parity of # true incident edges = label



$$\begin{array}{rr} (x \vee y) & \wedge\ (\overline{x} \vee z) \\ \wedge\ (\overline{x} \vee \overline{y}) & \wedge\ (y \vee \overline{z}) \\ \wedge\ (x \vee \overline{z}) & \wedge\ (\overline{y} \vee z) \end{array}$$

Requires proof size $\exp(\Omega(N))$ on well-connected so-called expander graphs — **"resolution cannot count** $\mathrm{mod}\ 2$**"**

Examples of Hard Formulas for Resolution (3/3)

Random k-CNF formulas [Chvátal & Szemerédi '88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

Examples of Hard Formulas for Resolution (3/3)

Random k-CNF formulas [Chvátal & Szemerédi '88]
Δn randomly sampled k-clauses over n variables

($\Delta \gtrsim 4.5$ sufficient to get unsatisfiable 3-CNF almost surely)

Again lower bound $\exp(\Omega(N))$

And more...

- k-colourability [Beame et al. '05]
- Independent sets and vertex covers [Beame et al. '07]
- Zero-one designs [Mikša & Nordström '14]
- Et cetera...

Theoretical Lower Bounds and Practical Reality

- So resolution is very weak in theory
- Then how can SAT solvers based on resolution be so good?
- One answer: this kind of formulas don't show up too often in practice
- Another area of intense research: Try to describe what properties of "real-life" formulas make them easy or hard
- But sometimes we would like to solve such formulas (and frankly they don't seem too hard, do they?)
- Can we go beyond resolution?

Polynomial Calculus (1/2)

Introduced in [Clegg et al. '96] and [Alekhnovich et al. '02]

Clauses translated to polynomial equations
Example: $x \vee y \vee \overline{z}$ gets translated to $xy\overline{z} = 0$
(Think of $0 \equiv \mathit{true}$ and $1 \equiv \mathit{false}$)

Compute only with 0 and 1: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 0$

Polynomial Calculus (1/2)

Introduced in [Clegg et al. '96] and [Alekhnovich et al. '02]

Clauses translated to polynomial equations
Example: $x \lor y \lor \overline{z}$ gets translated to $xy\overline{z} = 0$
(Think of $0 \equiv \mathit{true}$ and $1 \equiv \mathit{false}$)

Compute only with 0 and 1: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 1 = 0$

Derivation rules

Boolean axioms $\dfrac{}{x^2 - x = 0}$ Negation $\dfrac{}{x + \overline{x} = 1}$

Linear combination $\dfrac{p = 0 \quad q = 0}{\alpha p + \beta q = 0}$ Multiplication $\dfrac{p = 0}{xp = 0}$

Polynomial Calculus (2/2)

- Translate all clauses to polynomial equations
- Apply derivation rules
- Derive $1 = 0$ $\Leftrightarrow$ no common root $\Leftrightarrow$ formula unsatisfiable
- Also makes sense to do this for general polynomial equations (not translations of CNF formulas)
- Known as Gröbner basis computations (now you know the buzzword)
- Exponentially more powerful than resolution (e.g. Tseitin formulas easy)

Cutting Planes (1/2)

Introduced in [Cook et al. '87]

Clauses translated to linear inequalities over the reals with integer coefficients

Example: $x \lor y \lor \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$
or equivalently $x + y - z \geq 0$
(Now $1 \equiv \mathit{true}$ and $0 \equiv \mathit{false}$ again)

Cutting Planes (1/2)

Introduced in [Cook et al. '87]

Clauses translated to linear inequalities over the reals with integer coefficients

Example: $x \lor y \lor \overline{z}$ gets translated to $x + y + (1 - z) \geq 1$
or equivalently $x + y - z \geq 0$
(Now $1 \equiv \mathit{true}$ and $0 \equiv \mathit{false}$ again)

Derivation rules

Variable axioms $\dfrac{}{0 \leq x \leq 1}$

Multiplication $\dfrac{\sum a_i x_i \geq A}{\sum c a_i x_i \geq cA}$

Addition $\dfrac{\sum a_i x_i \geq A \quad \sum b_i x_i \geq B}{\sum (a_i + b_i) x_i \geq A + B}$

Division $\dfrac{\sum c a_i x_i \geq A}{\sum a_i x_i \geq \lceil A/c \rceil}$

Cutting Planes (2/2)

- Translate all clauses to linear inequalities
- Apply derivation rules
- Derive $0 \geq 1 \Leftrightarrow$ formula unsatisfiable
- Also makes sense for more general linear inequalities (not translations of CNF formulas)

Cutting Planes (2/2)

- Translate all clauses to linear inequalities
- Apply derivation rules
- Derive $0 \geq 1$ $\Leftrightarrow$ formula unsatisfiable
- Also makes sense for more general linear inequalities (not translations of CNF formulas)

Cutting planes

- can always simulate resolution proofs efficiently
- is sometimes exponentially stronger (e.g., for PHP formulas just count to see #pigeons $>$ #holes)

Algebraic or Geometric SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [Clegg et al. '96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus

Algebraic or Geometric SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [Clegg et al. '96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- More work has been done on so-called pseudo-Boolean solvers using (subset of) cutting planes reasoning
- But again seems hard to make competitive with CDCL

Algebraic or Geometric SAT Solvers?

- Quite some excitement about Gröbner basis approach to SAT solving after [Clegg et al. '96]
- Promise of performance improvement failed to deliver
- Meanwhile: the CDCL revolution in late 1990s...
- Some current SAT solvers do Gaussian elimination, but this is only very limited form of polynomial calculus
- More work has been done on so-called pseudo-Boolean solvers using (subset of) cutting planes reasoning
- But again seems hard to make competitive with CDCL
- Is it harder to build good algebraic or geometric SAT solvers? Or is it just that too little work has been done? (Or both?)

So... Is There a Smarter Way Than Brute-Force?

In theory, probably no...

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct in the worst case
- Proving (or disproving) this is one of Millennium Prize Problems: Are there efficient algorithms for NP-problems?

So... Is There a Smarter Way Than Brute-Force?

In theory, probably no...

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct in the worst case
- Proving (or disproving) this is one of Millennium Prize Problems: Are there efficient algorithms for NP-problems?

In practice, definitely yes!

- Real-world problems are usually not "worst-case" but highly structured
- Fairly simple (but clever) methods work amazingly well amazingly often (though we don't really understand why)

So... Is There a Smarter Way Than Brute-Force?

In theory, probably no...

- COLOURING, CLIQUE, SAT, and 1000s other problems are "all the same" — efficient algorithm for one can solve all
- Widely believed impossible to construct algorithms that are always (a) efficient and (b) correct in the worst case
- Proving (or disproving) this is one of Millennium Prize Problems: Are there efficient algorithms for NP-problems?

In practice, definitely yes!

- Real-world problems are usually not "worst-case" but highly structured
- Fairly simple (but clever) methods work amazingly well amazingly often (though we don't really understand why)

Stark disconnect between theory and practice...

Our Research Goals

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

Our Research Goals

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

Construct stronger algorithms for combinatorial problems

- Use insights into stronger mathematical methods of reasoning to build algorithms for SAT and related problems
- Aiming for exponential speed-ups over state of the art

Our Research Goals

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

Construct stronger algorithms for combinatorial problems

- Use insights into stronger mathematical methods of reasoning to build algorithms for SAT and related problems
- Aiming for exponential speed-ups over state of the art

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

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 — also suggests 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 — also suggests stronger methods of reasoning
- Lots of challenging work for PhD students (we're hiring!)
- Also room for interesting BSc or MSc projects

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 — also suggests stronger methods of reasoning
- Lots of challenging work for PhD students (we're hiring!)
- Also room for interesting BSc or MSc projects

Thank you for your attention!