On Proof Complexity Lower Bounds and Possible Connections to SAT Solving

Jakob Nordström

KTH Royal Institute of Technology
Stockholm, Sweden

Mathematical Logic: Proof Theory, Constructive Mathematics
Mathematisches Forschungsinstitut Oberwolfach
November 6–12, 2011

Based on joint work with Eli Ben-Sasson

A Fundamental Theoretical Problem…

Problem

Given a propositional logic formula F, can we decide efficiently whether it is true no matter how we assign values to its variables?

TAUTOLOGY: Fundamental problem in theoretical computer science ever since Stephen Cook's NP-completeness paper in 1971

(And significance realized much earlier — cf. Gödel's letter in 1956)

These days recognized as one of the main challenges for all of mathematics — one of the million dollar "Millennium Problems"

A Fundamental Theoretical Problem...

Problem

Given a propositional logic formula F, can we decide efficiently whether it is true no matter how we assign values to its variables?

TAUTOLOGY: Fundamental problem in theoretical computer science ever since Stephen Cook's NP-completeness paper in 1971

(And significance realized much earlier — cf. Gödel's letter in 1956)

These days recognized as one of the main challenges for all of mathematics — one of the million dollar "Millennium Problems"

. . . with Huge Practical Implications

- All known algorithms run in exponential time in worst case
- But enormous progress on applied computer programs last 10-15 years
- These so-called SAT solvers are routinely deployed to solve large-scale real-world problems with millions of variables
- Used in e.g. hardware verification, software testing, software package management, artificial intelligence, cryptography, bioinformatics, and more
- But also exist small example formulas with only hundreds of variables that trip up even state-of-the-art SAT solvers

What Makes Formulas Hard or Easy?

- Best algorithms today based on simple DPLL method (Davis-Putnam-Logemann-Loveland) from 1960s (although with many clever optimizations)
- Corresponds to search algorithm for resolution proof system
- How can these SAT solvers be so good in practice? And how can one know whether a particular formula is tractable or too difficult?
- This talk: What can (lower bounds in) proof complexity say about these questions?

Tautologies and CNF Formulas

Conjunctive normal form (CNF)

ANDs of ORs of variables or negated variables
(or conjunctions of disjunctive clauses)

Example:

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

Proving that a formula in propositional logic is **always** satisfied

$\Updownarrow$

Proving that a CNF formula is **never** satisfied
(i.e., evaluates to false however you set the variables)

Tautologies and CNF Formulas

Conjunctive normal form (CNF)

ANDs of ORs of variables or negated variables
(or conjunctions of disjunctive clauses)

Example:

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

Proving that a formula in propositional logic is **always** satisfied

$\Updownarrow$

Proving that a CNF formula is **never** satisfied
(i.e., evaluates to false however you set the variables)

Some Terminology

- Literal a: variable x or its negation $\overline{x}$
- Clause $C = a_1 \vee \cdots \vee a_k$: disjunction of literals
- CNF formula $F = C_1 \wedge \cdots \wedge C_m$: conjunction of clauses
- k-CNF formula: CNF formula with clauses of size $\leq k$
- All formulas k-CNFs in this talk (for arbitrary but fixed k)

The DPLL Method

Based on [Davis & Putnam '60] and [Davis, Logemann & Loveland '62]

Somewhat simplified description:

- If F contains an empty clause (without literals), then report "unsatisfiable"
- Otherwise pick some variable x in F
- Set $x = 0$, simplify F and try to refute recursively
- Set $x = 1$, simplify F and try to refute recursively
- If both cases result in "unsatisfiable", then report "unsatisfiable"

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

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

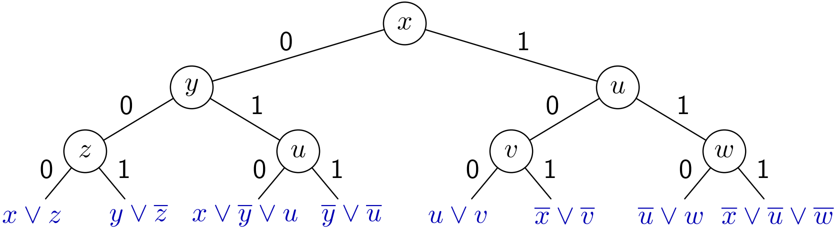
Pick variables in internal nodes; terminate in leaves when falsfied clause found

A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

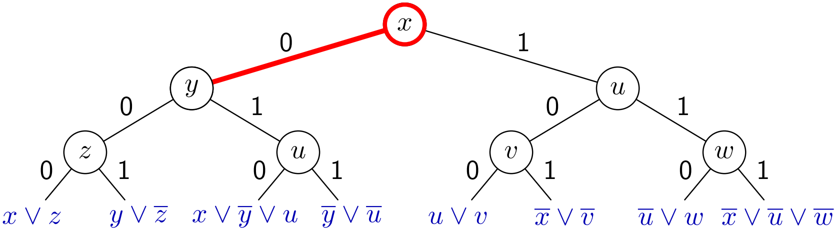


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

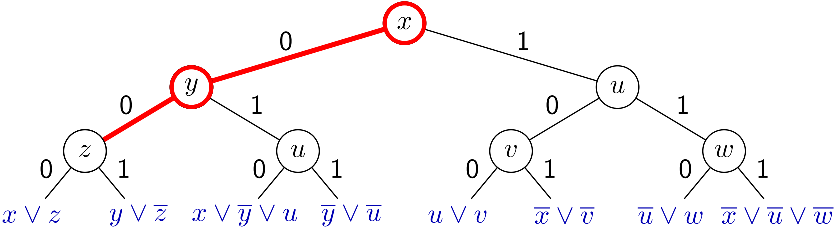


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

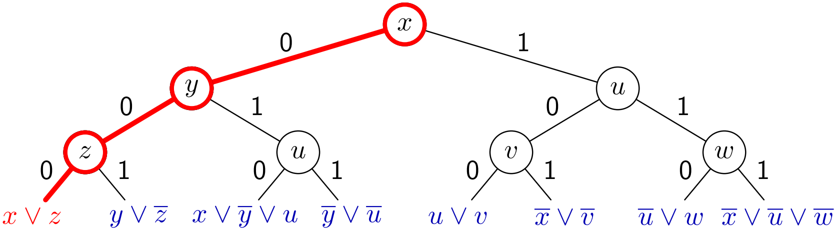


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

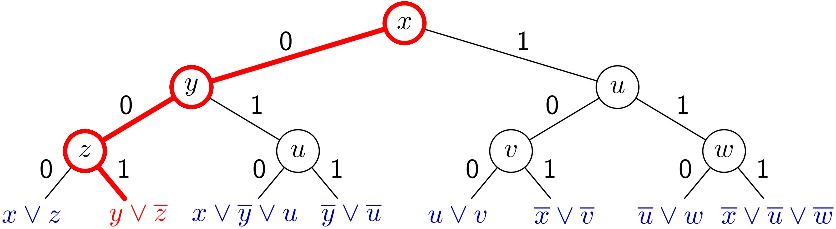


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

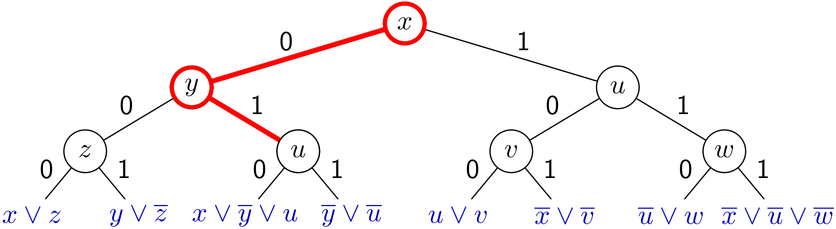


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

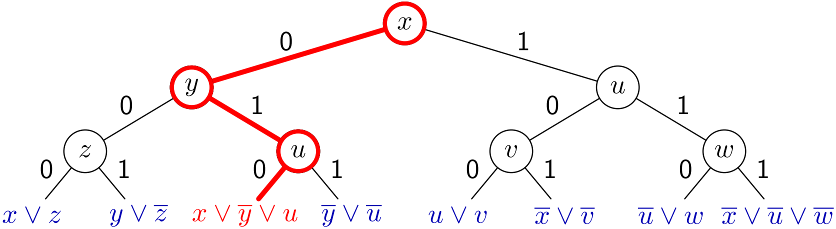


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

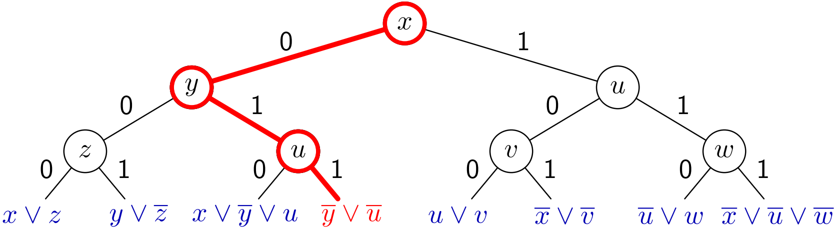


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found

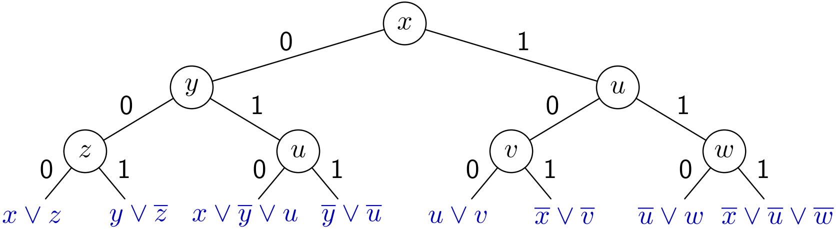


A DPLL Toy Example

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

Visualize execution of DPLL algorithm as search tree

Pick variables in internal nodes; terminate in leaves when falsfied clause found



State-of-the-art DPLL SAT solvers

Many more ingredients in modern SAT solvers, for instance:

- Choice of pivot variables crucial
- When reaching falsified clause, compute why partial assignment failed — add this info to formula as new clause (clause learning)
- Every once in a while, restart from beginning (but save computed info)

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.

Prove F unsatisfiable by deriving the unsatisfiable empty clause 0 from F by resolution

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.

Prove F unsatisfiable by deriving the unsatisfiable empty clause 0 from F by resolution

Resolution

Resolution rule:

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

Observation

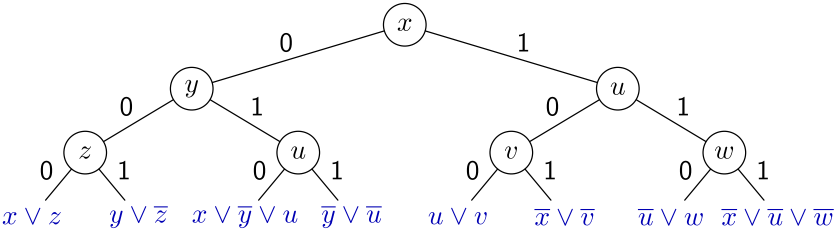
If F is a satisfiable CNF formula and D is derived from clauses $C_1, C_2 \in F$ by the resolution rule, then $F \wedge D$ is satisfiable.

Prove F unsatisfiable by deriving the unsatisfiable empty clause 0 from F by resolution

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again

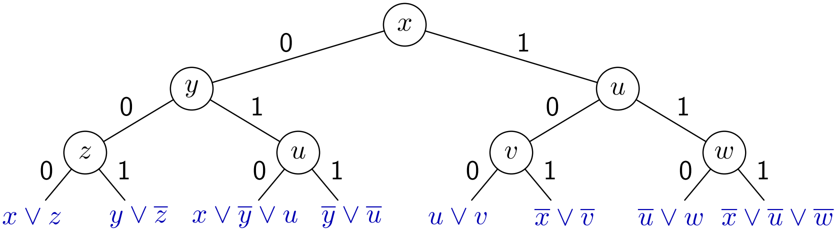


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again

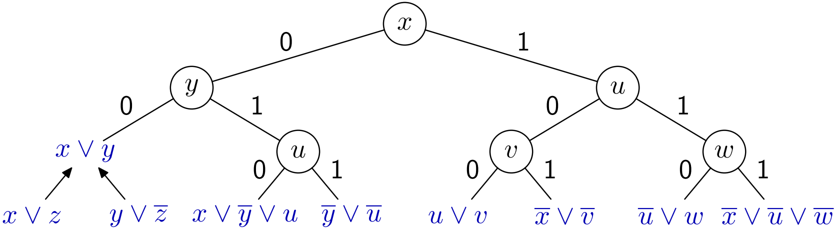


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again

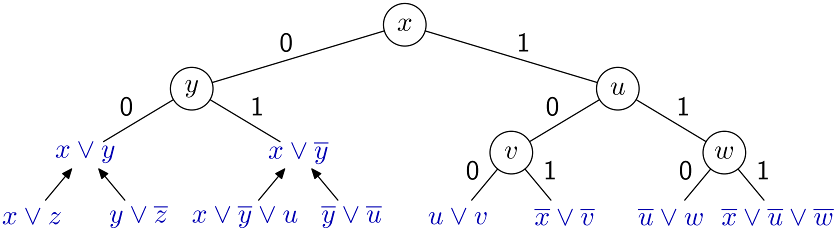


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again

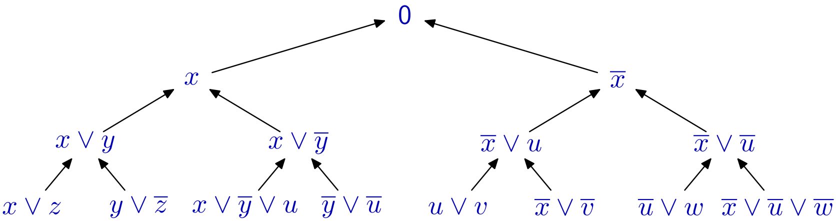


and apply resolution rule bottom-up

DPLL and Resolution

A DPLL execution is essentially a resolution proof

Look at our example again



and apply resolution rule bottom-up

Complexity Measures for Resolution

Let n = size of formula

Length

clauses in refutation — at most $\exp(n)$

Width

Size of largest clause in refutation — at most n

Space

Max # clauses one needs to remember when "verifying correctness of refutation on blackboard" — at most n (!)

Length

- Clearly lower bound on running time for any DPLL algorithm
- But if there is a short refutation, not clear how to find it
- In fact, probably intractable [Aleknovich & Razborov '01]
- So small length upper bound might be much too optimistic
- Not the right measure of "hardness in practice"

Length

- Clearly lower bound on running time for any DPLL algorithm
- But if there is a short refutation, not clear how to find it
- In fact, probably intractable [Aleknovich & Razborov '01]
- So small length upper bound might be much too optimistic
- Not the right measure of "hardness in practice"

Length

- Clearly lower bound on running time for any DPLL algorithm
- But if there is a short refutation, not clear how to find it
- In fact, probably intractable [Aleknovich & Razborov ’01]
- So small length upper bound might be much too optimistic
- Not the right measure of “hardness in practice”

Length

- Clearly lower bound on running time for any DPLL algorithm
- But if there is a short refutation, not clear how to find it
- In fact, probably intractable [Aleknovich & Razborov '01]
- So small length upper bound might be much too optimistic
- Not the right measure of "hardness in practice"

Length

- Clearly lower bound on running time for any DPLL algorithm
- But if there is a short refutation, not clear how to find it
- In fact, probably intractable [Aleknovich & Razborov '01]
- So small length upper bound might be much too optimistic
- Not the right measure of "hardness in practice"

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width ⇒ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width ⇒ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width $\Rightarrow$ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width $\Rightarrow$ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width $\Rightarrow$ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width $\Rightarrow$ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width $\Rightarrow$ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width $\Rightarrow$ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width $\Rightarrow$ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width $\Rightarrow$ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Length vs. Width

- Searching for small width refutations known heuristic in AI community
- Small width $\Rightarrow$ small length (by counting)
- But small length does not necessary imply small width — can have $\sqrt{n}$ width and linear length [Bonet & Galesi '99]
- However, really large (e.g., linear) width implies really large (exponential) length [Ben-Sasson & Wigderson '99]
- Small width $\Rightarrow$ DPLL solver will provably be fast [Atserias et al. '09] (but slighly idealized theoretical model)
- Right hardness measure?

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for “usual suspects” formulas in ’99, ’00, ’01. . . — always coincided with width bounds!?
- [Atserias & Dalmau ’03]: Space $\geq$ width (proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. ’06], [N. & Håstad ’08], [Ben-Sasson & N. ’08]: Upper bounds on width don’t say anything about space (but space model arguably **very** idealized and theoretical)

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for "usual suspects" formulas in '99, '00, '01... — always coincided with width bounds!?
- [Atserias & Dalmau '03]: Space $\geq$ width (proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. '06], [N. & Håstad '08], [Ben-Sasson & N. '08]: Upper bounds on width don't say anything about space (but space model arguably **very** idealized and theoretical)

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for "usual suspects" formulas in '99, '00, '01... — always coincided with width bounds!?
- [Atserias & Dalmau '03]: Space $\geq$ width (proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. '06], [N. & Håstad '08], [Ben-Sasson & N. '08]: Upper bounds on width don't say anything about space (but space model arguably **very** idealized and theoretical)

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for "usual suspects" formulas in '99, '00, '01... — always coincided with width bounds!?
- [Atserias & Dalmau '03]: Space $\geq$ width (proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. '06], [N. & Håstad '08], [Ben-Sasson & N. '08]: Upper bounds on width don't say anything about space (but space model arguably **very** idealized and theoretical)

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for "usual suspects" formulas in '99, '00, '01. . . — always coincided with width bounds!?
- [Atserias & Dalmau '03]: Space $\geq$ width
(proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. '06], [N. & Håstad '08], [Ben-Sasson & N. '08]:
Upper bounds on width don't say anything about space
(but space model arguably **very** idealized and theoretical)

Width vs. Space

- In practice, memory consumption is a very important bottleneck for SAT solvers
- So maybe space complexity can be relevant hardness measure?
- Sequence of lower bound results for “usual suspects” formulas in ’99, ’00, ’01... — always coincided with width bounds!?
- [Atserias & Dalmau ’03]: Space $\geq$ width
  (proven via Ehrenfeucht-Fraïssé games in finite model theory)
- But are space and width somehow the same measure or different?
- [N. ’06], [N. & Håstad ’08], [Ben-Sasson & N. ’08]:
  Upper bounds on width don’t say anything about space
  (but space model arguably **very** idealized and theoretical)

Our Results (Slightly) More Formally

Theorem (Ben-Sasson & N., FOCS '08)

There are k-CNF formula families of size $\mathcal{O}(n)$ with

- *refutation length* $\mathcal{O}(n)$
- *refutation width* $\mathcal{O}(1)$
- *refutation space* $\Omega(n/\log n)$.

Theorem (Ben-Sasson & N., ICS '11)

There are k-CNF formula families which are

- *very easy w.r.t. length (but then space large),*
- *very easy w.r.t. space (but then length large),*
- *any meaningful simultaneous optimization impossible.*

Our Results (Slightly) More Formally

Theorem (Ben-Sasson & N., FOCS '08)

There are k-CNF formula families of size $\mathcal{O}(n)$ with

- *refutation length $\mathcal{O}(n)$*
- *refutation width $\mathcal{O}(1)$*
- *refutation space $\Omega(n/\log n)$.*

Theorem (Ben-Sasson & N., ICS '11)

There are k-CNF formula families which are

- *very easy w.r.t. length (but then space large),*
- *very easy w.r.t. space (but then length large),*
- *any meaningful simultaneous optimization impossible.*

How to Get a Handle on Time-Space Relations?

Time-space trade-off questions well-studied for pebble games modelling calculations described by directed acyclic graphs ([Cook & Sethi '76] and many others)

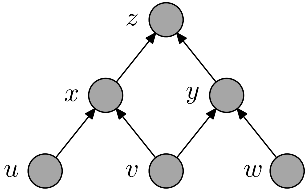
- Time needed for calculation: # pebbling moves
- Space needed for calculation: max # pebbles required

Some quick graph terminology

- DAGs consist of vertices with directed edges between them
- vertices with no incoming edges: sources
- vertices with no outgoing edges: sinks

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

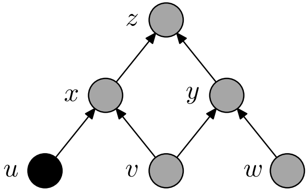


# moves	0
Current # pebbles	0
Max # pebbles so far	0

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

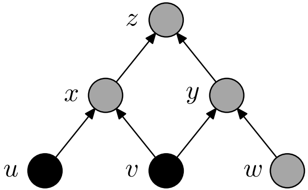


# moves	1
Current # pebbles	1
Max # pebbles so far	1

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

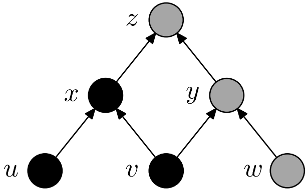


# moves	2
Current # pebbles	2
Max # pebbles so far	2

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

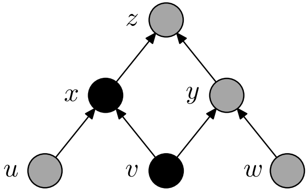


# moves	3
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

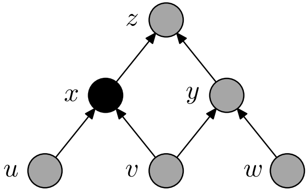


# moves	4
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

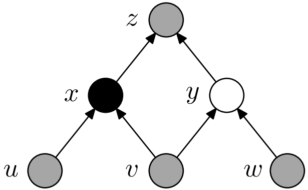


# moves	5
Current # pebbles	1
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

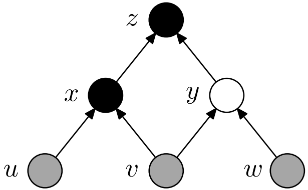


# moves	6
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

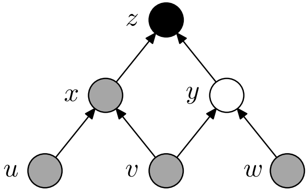


# moves	7
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

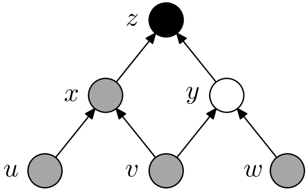


# moves	8
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of *G*

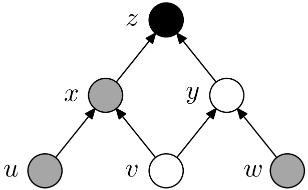


# moves	8
Current # pebbles	2
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex *v* if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from *v* if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

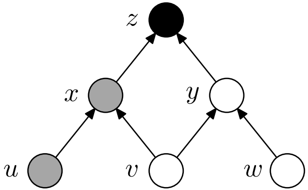


# moves	9
Current # pebbles	3
Max # pebbles so far	3

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of *G*

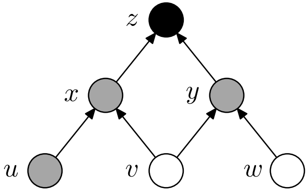


# moves	10
Current # pebbles	4
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex *v* if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from *v* if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

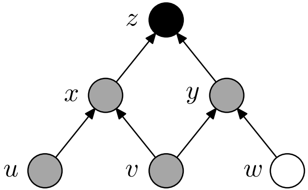


# moves	11
Current # pebbles	3
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G

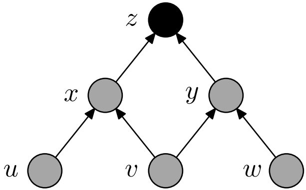


# moves	12
Current # pebbles	2
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

The Black-White Pebble Game

Goal: get single black pebble on sink vertex of G



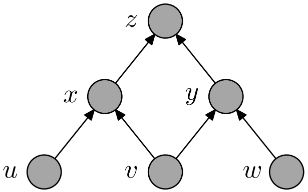
# moves	13
Current # pebbles	1
Max # pebbles so far	4

1. Can place black pebble on (empty) vertex v if all immediate predecessors have pebbles on them
2. Can always remove black pebble from vertex
3. Can always place white pebble on (empty) vertex
4. Can remove white pebble from v if all immediate predecessors have pebbles on them

Pebbling Contradiction

CNF formula encoding pebble game on DAG G

1. u
2. v
3. w
4. $\overline{u} \vee \overline{v} \vee x$
5. $\overline{v} \vee \overline{w} \vee y$
6. $\overline{x} \vee \overline{y} \vee z$
7. $\overline{z}$



- sources are true
- truth propagates upwards
- but sink is false

Studied by [Bonet et al. ’98, Raz & McKenzie ’99, Ben-Sasson & Wigderson ’99] and others

Resolution–Pebbling Correspondence

Observation (Ben-Sasson et al. '00)

Any black-pebbles-only pebbling translates into refutation with

- *refutation length $\leq$ # moves*
- *space $\leq$ # pebbles*

Theorem (Ben-Sasson '02)

Any refutation translates into black-white pebbling with

- *# moves $\leq$ refutation length*
- *# pebbles $\leq$ # variables mentioned simultaneously in refutation*

Unfortunately extremely easy w.r.t. space! (counting clauses)

Resolution–Pebbling Correspondence

Observation (Ben-Sasson et al. '00)

Any black-pebbles-only pebbling translates into refutation with

- *refutation length $\leq$ # moves*
- *space $\leq$ # pebbles*

Theorem (Ben-Sasson '02)

Any refutation translates into black-white pebbling with

- *# moves $\leq$ refutation length*
- *# pebbles $\leq$ # variables mentioned simultaneously in refutation*

Unfortunately extremely easy w.r.t. space! (counting clauses)

Resolution–Pebbling Correspondence

Observation (Ben-Sasson et al. '00)

Any black-pebbles-only pebbling translates into refutation with

- *refutation length $\leq$ # moves*
- *space $\leq$ # pebbles*

Theorem (Ben-Sasson '02)

Any refutation translates into black-white pebbling with

- *# moves $\leq$ refutation length*
- *# pebbles $\leq$ # variables mentioned simultaneously in refutation*

Unfortunately extremely easy w.r.t. space! (counting clauses)

Key Idea: Variable Substitution

Make formula harder by substituting $x_1 \oplus x_2$ for every variable x (also works for other Boolean functions with “right” properties):

$$\overline{x} \vee y$$

$$\Downarrow$$

$$\neg(x_1 \oplus x_2) \vee (y_1 \oplus y_2)$$

$$\Downarrow$$

$$\begin{aligned} & (x_1 \vee \overline{x}_2 \vee y_1 \vee y_2) \\ \wedge\ & (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee y_1 \vee y_2) \\ \wedge\ & (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2) \end{aligned}$$

Key Technical Result: Substitution Theorem

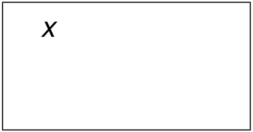
Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F



Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

x
$\overline{x} \vee y$

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

x
$\overline{x} \lor y$
y

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

x
$\overline{x} \lor y$
y

$x_1 \lor x_2$
$\overline{x}_1 \lor \overline{x}_2$

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

F	$F[\oplus]$
x	$x_1 \lor x_2$
	$\overline{x}_1 \lor \overline{x}_2$
$\overline{x} \lor y$	$x_1 \lor \overline{x}_2 \lor y_1 \lor y_2$
	$x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor \overline{y}_2$
	$\overline{x}_1 \lor x_2 \lor y_1 \lor y_2$
	$\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor \overline{y}_2$
y	

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

$$x$$
$$\overline{x} \lor y$$
$$y$$

$$x_1 \lor x_2$$
$$\overline{x}_1 \lor \overline{x}_2$$
$$x_1 \lor \overline{x}_2 \lor y_1 \lor y_2$$
$$x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor \overline{y}_2$$
$$\overline{x}_1 \lor x_2 \lor y_1 \lor y_2$$
$$\overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor \overline{y}_2$$
$$y_1 \lor y_2$$
$$\overline{y}_1 \lor \overline{y}_2$$

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

$$
\boxed{\begin{array}{l} x \\ \overline{x} \lor y \\ y \end{array}}
$$

$$
\boxed{\begin{array}{l} x_1 \lor x_2 \\ \overline{x}_1 \lor \overline{x}_2 \\ x_1 \lor \overline{x}_2 \lor y_1 \lor y_2 \\ x_1 \lor \overline{x}_2 \lor \overline{y}_1 \lor \overline{y}_2 \\ \overline{x}_1 \lor x_2 \lor y_1 \lor y_2 \\ \overline{x}_1 \lor x_2 \lor \overline{y}_1 \lor \overline{y}_2 \\ y_1 \lor y_2 \\ \overline{y}_1 \lor \overline{y}_2 \end{array}}
$$

For such refutation of $F[\oplus]$:

- length $\geq$ length for F
- space $\geq$ # variables simultaneously for F

Key Technical Result: Substitution Theorem

Let $F[\oplus]$ denote formula with XOR $x_1 \oplus x_2$ substituted for x

Obvious approach for refuting $F[\oplus]$: mimic refutation of F

$$\boxed{\begin{array}{l} x \\ \overline{x} \vee y \\ y \end{array}}$$

$$\boxed{\begin{array}{l} x_1 \vee x_2 \\ \overline{x}_1 \vee \overline{x}_2 \\ x_1 \vee \overline{x}_2 \vee y_1 \vee y_2 \\ x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2 \\ \overline{x}_1 \vee x_2 \vee y_1 \vee y_2 \\ \overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2 \\ y_1 \vee y_2 \\ \overline{y}_1 \vee \overline{y}_2 \end{array}}$$

For such refutation of $F[\oplus]$:

- length $\geq$ length for F
- space $\geq$ # variables
  simultaneously for F

Prove that this is (sort of) best one can do for $F[\oplus]$!

Pieces Together: Substitution + Pebbling Formulas

Making variable substitutions in pebbling formulas

- lifts lower bound from # variables to # clauses (i.e., space)
- maintains upper bound in terms of space and length

Get our results by

- using known pebbling results from literature of 70s and 80s
- proving a couple of new pebbling results [N. '10]
- to get tight trade-offs, showing that resolution can sometimes do better than black-only pebbling [N. '10]

Pieces Together: Substitution + Pebbling Formulas

Making variable substitutions in pebbling formulas

- lifts lower bound from # variables to # clauses (i.e., space)
- maintains upper bound in terms of space and length

Get our results by

- using known pebbling results from literature of 70s and 80s
- proving a couple of new pebbling results [N. '10]
- to get tight trade-offs, showing that resolution can sometimes do better than black-only pebbling [N. '10]

Extending the Results to Stronger Proof Systems?

Key technical theorem: amplify space lower bounds through variable substitution

Almost completely oblivious to which proof system is being studied

Extended to strictly stronger k-DNF resolution proof systems — maybe can be made to work for other stronger systems as well?

Open Question

Can the Substitution Theorem be proven for, say, Cutting Planes or Polynomial Calculus (with/without Resolution), thus yielding time-space trade-offs for these proof systems as well?

Extending the Results to Stronger Proof Systems?

Key technical theorem: amplify space lower bounds through variable substitution

Almost completely oblivious to which proof system is being studied

Extended to strictly stronger k-DNF resolution proof systems — maybe can be made to work for other stronger systems as well?

Open Question

Can the Substitution Theorem be proven for, say, Cutting Planes or Polynomial Calculus (with/without Resolution), thus yielding time-space trade-offs for these proof systems as well?

Some Other Open Theoretical Problems

- Many more open (theoretical) questions about length, width, and space in proof complexity
- See recent survey *Pebble Games, Proof Complexity, and Time-Space Trade-offs* at my webpage for details
- To conclude this talk, want to focus on main applied question

Is Tractability Captured by Space Complexity?

Open Question

Do our space lower bounds and trade-offs imply anything "in real life" for state-of-the-art SAT solvers?

That is, does space complexity capture hardness?

Preliminary experiments indicate that pebbling formulas with high space complexity might be hard in practice for SAT solvers

Note that pebbling formulas always extremely easy with respect to length and width, so hardness in practice would be intriguing

Is Tractability Captured by Space Complexity?

Open Question

Do our space lower bounds and trade-offs imply anything “in real life” for state-of-the-art SAT solvers?

That is, does space complexity capture hardness?

Preliminary experiments indicate that pebbling formulas with high space complexity might be hard in practice for SAT solvers

Note that pebbling formulas always extremely easy with respect to length and width, so hardness in practice would be intriguing

Is Tractability Captured by Space Complexity?

Open Question

Do our space lower bounds and trade-offs imply anything "in real life" for state-of-the-art SAT solvers?

That is, does space complexity capture hardness?

Preliminary experiments indicate that pebbling formulas with high space complexity might be hard in practice for SAT solvers

Note that pebbling formulas always extremely easy with respect to length and width, so hardness in practice would be intriguing

Take-Home Message

- Modern SAT solvers, although based on old and simple DPLL method, can be enormously successful in practice
- Key issue is to minimize time and memory consumption
- However, our results suggest strong time-space trade-offs that should make this impossible
- Many remaining open questions about space in proof complexity
- Main open practical question: is tractability captured by space complexity?

Thank you for your attention!