Seeking Practical CDCL Insights from Theoretical SAT Benchmarks

Jakob Nordström

KTH Royal Institute of Technology
Stockholm, Sweden

7th Pragmatics of SAT Workshop
Bordeaux, France
July 4, 2016

Joint work with Jan Elffers, Karem Sakallah, and Laurent Simon

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]
- Addition of conflict-driven clause learning (CDCL) [MS99]
  exponential increase in reasoning power

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]
- Addition of conflict-driven clause learning (CDCL) [MS99] exponential increase in reasoning power
- Plus lots of smart engineering to make it fly in practice [MMZ+01]

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]
- Addition of conflict-driven clause learning (CDCL) [MS99]
  exponential increase in reasoning power
- Plus lots of smart engineering to make it fly in practice [MMZ$^+$01]
- And a sometimes somewhat bewildering alphabet soup of heuristics
  (VSIDS, 1UIP, LBD, BCD, BCE, BVA, ELS, FLP, VE, VMTF, ...)

The Unreasonable Effectiveness of SAT Solvers

- Huge improvements in SAT solving performance over last 15–20 years
- Basis of best modern SAT solvers still DPLL method [DP60, DLL62]
- Addition of conflict-driven clause learning (CDCL) [MS99] exponential increase in reasoning power
- Plus lots of smart engineering to make it fly in practice [MMZ+01]
- And a sometimes somewhat bewildering alphabet soup of heuristics (VSIDS, 1UIP, LBD, BCD, BCE, BVA, ELS, FLP, VE, VMTF, . . .)
- Want a **deeper understanding** of how these solvers actually work

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers
- Applied approach: Vary CDCL settings on industrial benchmarks
  Some work in [KSM11, SM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers
- Applied approach: Vary CDCL settings on industrial benchmarks
  Some work in [KSM11, SM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Why not combine the two approaches?

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers
- Applied approach: Vary CDCL settings on industrial benchmarks
  Some work in [KSM11, SM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Why not combine the two approaches?

- Generate scalable & easy versions of theoretical benchmarks
  Have short resolution proofs, so no excuse for solver not doing well...

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers
- Applied approach: Vary CDCL settings on industrial benchmarks
  Some work in [KSM11, SM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Why not combine the two approaches?

- Generate scalable & easy versions of theoretical benchmarks
  Have short resolution proofs, so no excuse for solver not doing well...
- Run CDCL with different heuristics to see how performance affected

Analysing Behaviour of CDCL Solvers

Can we explain when CDCL does well and when formulas are hard?
Run experiments and draw interesting conclusions?

- Theory approach: CDCL hardness related to complexity measures?
  Some work in [JMNŽ12], but generated more questions than answers
- Applied approach: Vary CDCL settings on industrial benchmarks
  Some work in [KSM11, SM11], but diversity and sparsity of industrial benchmarks makes it hard to draw clear conclusions

Why not combine the two approaches?

- Generate scalable & easy versions of theoretical benchmarks
  Have short resolution proofs, so no excuse for solver not doing well. . .
- Run CDCL with different heuristics to see how performance affected
- Benchmarks extremal w.r.t. different properties — can be expected to "challenge" solver

This Talk

- Describe candidate set of benchmarks
- Discuss CDCL parameter configurations to be tested
  (focus on basic CDCL search, not preprocessing techniques)
- Report on some preliminary findings
  Warning for sensitive viewers: will be plots, but no cactus plots
- **Caveat:** Still very much work in progress
  Hope that presentation can generate interesting discussions

Some Notation and Terminology

- Literal a: variable x or its negation $\overline{x}$ (or $\neg x$)
- 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
- k-CNF formula: CNF formula with clauses of size $\leq k$
  (where k is some constant)
- N denotes size of formula (# literals counted with repetitions)
- $\mathcal{O}(f(N))$ grows at most as quickly as $f(N)$ asymptotically
  $\Omega(g(N))$ grows at least as quickly as $g(N)$ asymptotically
  $\Theta(h(N))$ grows equally quickly as $h(N)$ asymptotically

Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

Proof ends when empty clause $\perp$ derived

Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

Proof ends when empty clause $\perp$ derived

1. $x \lor y$
2. $x \lor \overline{y} \lor z$
3. $\overline{x} \lor z$
4. $\overline{y} \lor \overline{z}$
5. $\overline{x} \lor \overline{z}$

Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

Proof ends when empty clause $\perp$ derived

Can represent proof/refutation as

- annotated list or
- directed acyclic graph

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

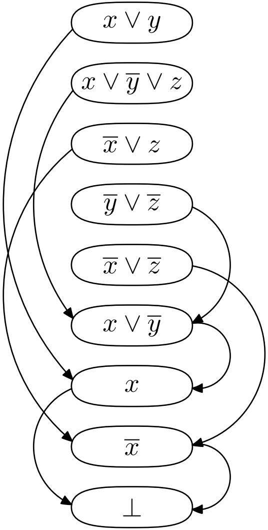
Derive new clauses by resolution rule

$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

Proof ends when empty clause $\perp$ derived

Can represent proof/refutation as

- annotated list or
- directed acyclic graph



Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

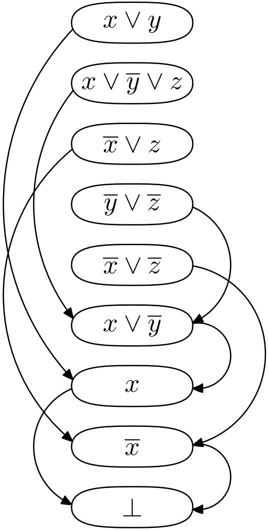
$$\frac{C \lor x \qquad D \lor \overline{x}}{C \lor D}$$

Proof ends when empty clause $\perp$ derived

Can represent proof/refutation as

- annotated list or
- directed acyclic graph

Tree-like if DAG is tree (corresponds to DPLL)



Proof System Underlying CDCL: Resolution

Goal: refute **unsatisfiable** CNF

Start with clauses of formula (axioms)

Derive new clauses by resolution rule

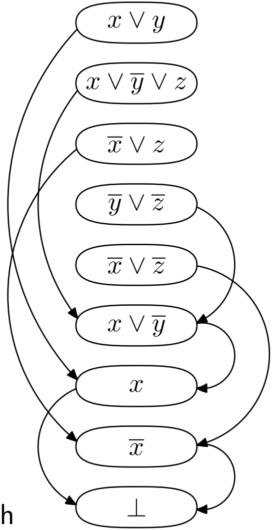
$$\frac{C \vee x \qquad D \vee \overline{x}}{C \vee D}$$

Proof ends when empty clause $\perp$ derived

Can represent proof/refutation as

- annotated list or
- directed acyclic graph

Tree-like if DAG is tree (corresponds to DPLL)
Regular if resolved variables don't repeat on path



Resolution Size/Length

Size/length of proof $=$ $\#$ clauses (9 in example on previous slide)
Length of refuting F $=$ min over all proofs for F

Resolution Size/Length

Size/length of proof = # clauses (9 in example on previous slide)
Length of refuting F = min over all proofs for F

Most fundamental measure in proof complexity

Lower bound on CDCL running time*
(can extract resolution proof from execution trace)

Never worse than $\exp(\mathcal{O}(N))$

Matching $\exp(\Omega(N))$ lower bounds known [Urq87, CS88, BW01]

Resolution Size/Length

Size/length of proof = # clauses (9 in example on previous slide)
Length of refuting F = min over all proofs for F

Most fundamental measure in proof complexity

Lower bound on CDCL running time*
(can extract resolution proof from execution trace)

Never worse than $\exp(\mathcal{O}(N))$

Matching $\exp(\Omega(N))$ lower bounds known [Urq87, CS88, BW01]

(*) Ignores preprocessing — focus here on CDCL proof search

Resolution Space

Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

1.	$x \lor y$	Axiom
2.	$x \lor \overline{y} \lor z$	Axiom
3.	$\overline{x} \lor z$	Axiom
4.	$\overline{y} \lor \overline{z}$	Axiom
5.	$\overline{x} \lor \overline{z}$	Axiom
6.	$x \lor \overline{y}$	Res$(2,4)$
7.	x	Res$(1,6)$
8.	$\overline{x}$	Res$(3,5)$
9.	$\perp$	Res$(7,8)$

Resolution Space

Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

Example: Space at step 7 . . .

1.	$x \vee y$	Axiom
2.	$x \vee \overline{y} \vee z$	Axiom
3.	$\overline{x} \vee z$	Axiom
4.	$\overline{y} \vee \overline{z}$	Axiom
5.	$\overline{x} \vee \overline{z}$	Axiom
6.	$x \vee \overline{y}$	Res$(2, 4)$
7.	x	Res$(1, 6)$
8.	$\overline{x}$	Res$(3, 5)$
9.	$\perp$	Res$(7, 8)$

Resolution Space

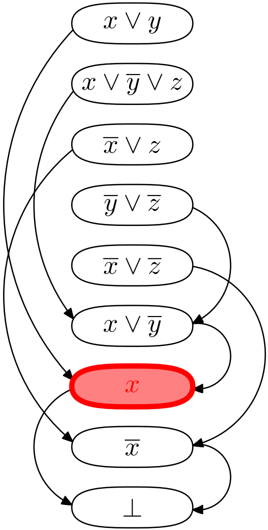
Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

Example: Space at step 7 . . .



Resolution Space

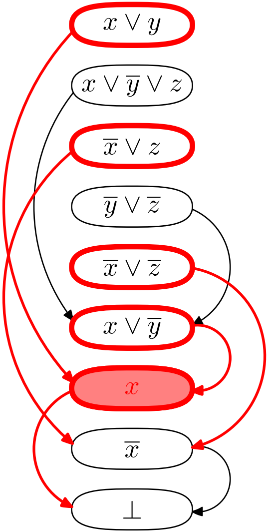
Space = max # clauses in memory when performing refutation

Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

Example: Space at step 7 is 5



Resolution Space

Space = max # clauses in memory when performing refutation

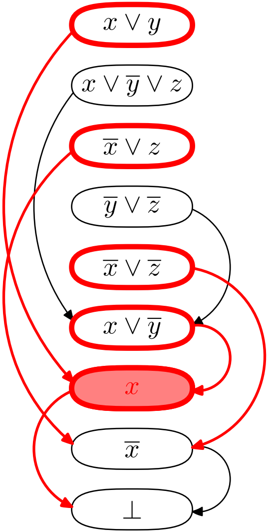
Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

Example: Space at step 7 is 5

Space of proof = max over all steps



Resolution Space

Space = max # clauses in memory when performing refutation

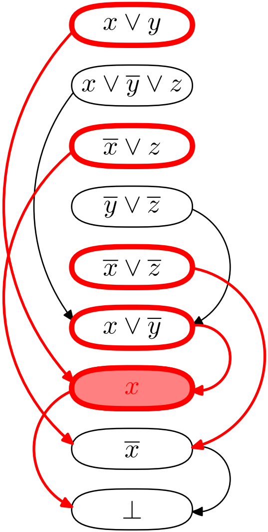
Motivated by SAT solver memory usage (but also intrinsically interesting for proof complexity)

Can be measured in different ways — makes most sense here to focus on clause space

Space at step t = # clauses at steps $\leq t$ used at steps $\geq t$

Example: Space at step 7 is 5

Space of proof = max over all steps
Space of refuting F = min over all proofs



Bounds on Resolution Space

Space always at most $N + \mathcal{O}(1)$ (!) [ET01]

Matching $\Omega(N)$ lower bounds known [ABRW02, BG03, ET01]

Bounds on Resolution Space

Space always at most $N + \mathcal{O}(1)$ (!) [ET01]

Matching $\Omega(N)$ lower bounds known [ABRW02, BG03, ET01]

Linear space lower bounds might not seem so impressive...

Bounds on Resolution Space

Space always at most $N + \mathcal{O}(1)$ (!) [ET01]

Matching $\Omega(N)$ lower bounds known [ABRW02, BG03, ET01]

Linear space lower bounds might not seem so impressive...

But:

- Hold even for optimal algorithms that magically know exactly which clauses to throw away or keep
- So significantly more space might be needed in practice
- And linear space upper bound obtained for proofs of exponential size

Resolution Width

Width of proof $=$ size of largest clause in proof (always $\leq N$)
Width of refuting F $=$ width of shortest proof for F

Resolution Width

Width of proof $=$ size of largest clause in proof (always $\leq N$)
Width of refuting F $=$ width of shortest proof for F

Width upper bounds $\Rightarrow$ length upper bounds (obvious)

Resolution Width

Width of proof $=$ size of largest clause in proof (always $\leq N$)
Width of refuting F $=$ width of shortest proof for F

Width upper bounds $\Rightarrow$ length upper bounds (obvious)

Width lower bounds $\Rightarrow$ space lower bounds [AD08]

Resolution Width

Width of proof $=$ size of largest clause in proof (always $\leq N$)
Width of refuting F $=$ width of shortest proof for F

Width upper bounds $\Rightarrow$ length upper bounds (obvious)

Width lower bounds $\Rightarrow$ space lower bounds [AD08]

Really strong width lower bounds $\Rightarrow$ length lower bounds [BW01]

Resolution Width

Width of proof $=$ size of largest clause in proof (always $\leq N$)
Width of refuting F $=$ width of shortest proof for F

Width upper bounds $\Rightarrow$ length upper bounds (obvious)

Width lower bounds $\Rightarrow$ space lower bounds [AD08]

Really strong width lower bounds $\Rightarrow$ length lower bounds [BW01]

But only moderately strong width lower bounds don't imply anything for length [BG01] (except hardness for tree-like resolution / DPLL)

Collection of Combinatorial Benchmarks

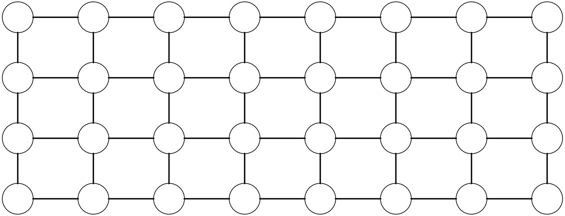
1. Tseitin formulas [Tse68, Urq87]
2. Ordering principle formulas [Kri85, Stå96]
3. Pebbling formulas [BW01, BN08]
4. Stone formulas [AJPU07]
5. Zero-one designs / subset cardinality formulas [Spe10, VS10, MN14]
6. Even colouring formulas [Mar06]
7. Relativized pigeonhole principle (RPHP) formulas [AMO13, ALN16]

Some General Comments on Benchmarks

- Tweak instances so that all have short resolution proofs (even linear size for all except relativized RPHP)
  - proofs can in principle be found by CDCL
  - without any preprocessing
  - often even without any restarts
  - sometimes even without learning, i.e., just DPLL (though might incur some blow-up)
  - ... given right variable decision order
- Test theoretical results in [AFT11, PD11]: Does CDCL search for proofs efficiently?
- Several benchmarks extremal w.r.t. proof complexity measures or trade-offs between measures (see workshop paper for details)
- Practical note: many (though not all) instances generated using **CNFgen** [CNF, LENV16]

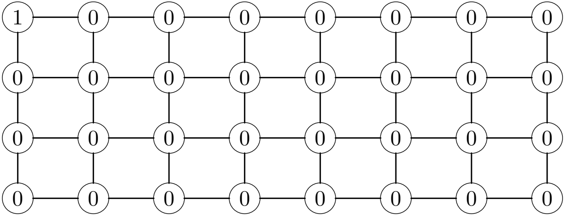
Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$



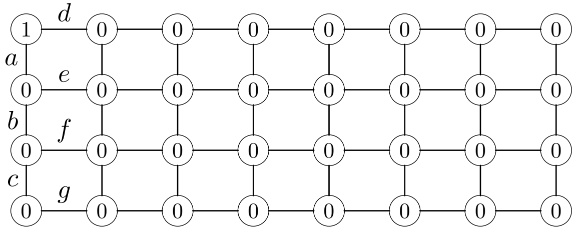
Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd



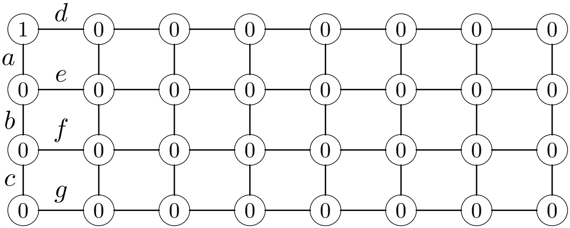
Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables = edges



Tseitin Formulas

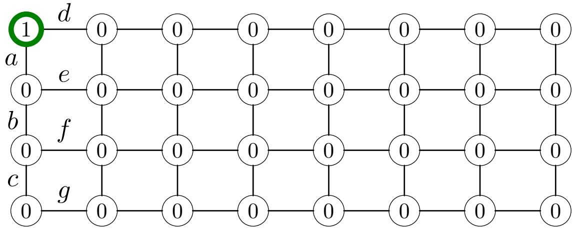
- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"



Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"

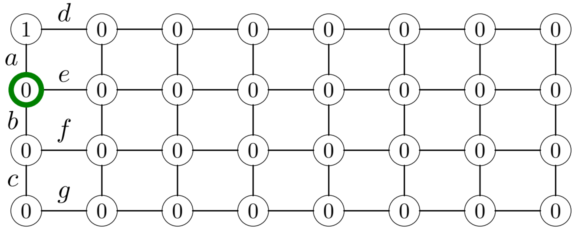
$$(a \lor d) \land (\overline{a} \lor \overline{d})$$



Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"

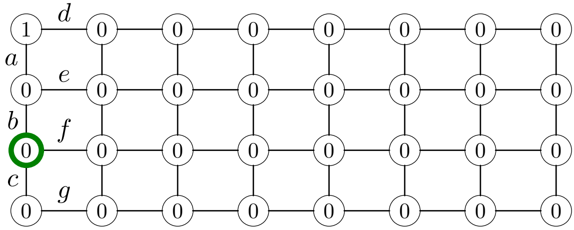
$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e})
\end{aligned}
$$



Tseitin Formulas

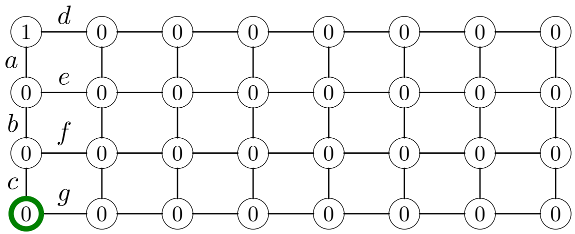
- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables $=$ edges
- Write down clauses encoding constraints
  "vertex label $=$ parity of incident edges"

$$
\begin{aligned}
& (a \vee d) \\
\wedge\ & (\overline{a} \vee \overline{d}) \\
\wedge\ & (a \vee b \vee \overline{e}) \\
\wedge\ & (a \vee \overline{b} \vee e) \\
\wedge\ & (\overline{a} \vee b \vee e) \\
\wedge\ & (\overline{a} \vee \overline{b} \vee \overline{e}) \\
\wedge\ & (b \vee c \vee \overline{f}) \\
\wedge\ & (b \vee \overline{c} \vee f) \\
\wedge\ & (\overline{b} \vee c \vee f) \\
\wedge\ & (\overline{b} \vee \overline{c} \vee \overline{f})
\end{aligned}
$$



Tseitin Formulas

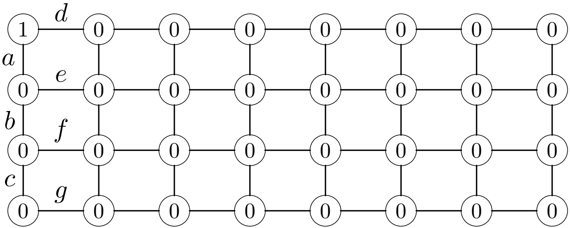
- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables = edges
- Write down clauses encoding constraints
  "vertex label = parity of incident edges"



$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e}) \\
\land\ & (b \lor c \lor \overline{f}) \\
\land\ & (b \lor \overline{c} \lor f) \\
\land\ & (\overline{b} \lor c \lor f) \\
\land\ & (\overline{b} \lor \overline{c} \lor \overline{f}) \\
\land\ & (c \lor \overline{g}) \\
\land\ & (\overline{c} \lor g)
\end{aligned}
$$

Tseitin Formulas

- Take $w \times m$ grid, $w \ll m$
- Label vertices $0/1$ so that total charge odd
- Let variables $=$ edges
- Write down clauses encoding constraints
  "vertex label $=$ parity of incident edges"
- Hard for well-connected graphs [Urq87] but easy on grids with $w = \mathcal{O}(1)$ (even for DPLL)



$$
\begin{aligned}
& (a \lor d) \\
\land\ & (\overline{a} \lor \overline{d}) \\
\land\ & (a \lor b \lor \overline{e}) \\
\land\ & (a \lor \overline{b} \lor e) \\
\land\ & (\overline{a} \lor b \lor e) \\
\land\ & (\overline{a} \lor \overline{b} \lor \overline{e}) \\
\land\ & (b \lor c \lor \overline{f}) \\
\land\ & (b \lor \overline{c} \lor f) \\
\land\ & (\overline{b} \lor c \lor f) \\
\land\ & (\overline{b} \lor \overline{c} \lor \overline{f}) \\
\land\ & (c \lor \overline{g}) \\
\land\ & (\overline{c} \lor g) \\
& \vdots
\end{aligned}
$$

Subset Cardinality Formulas / Zero-One Designs

Proposed by [Spe10, VS10]

Variables $=$ 1s in matrix with four 1s per row/column $+$ extra 1

Each row wants majority true; each column wants majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Subset Cardinality Formulas / Zero-One Designs

Proposed by [Spe10, VS10]

Variables = 1s in matrix with four 1s per row/column + extra 1

Each row wants majority true; each column wants majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\; & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\; & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\; & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\; & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\; & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\; & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\; & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Subset Cardinality Formulas / Zero-One Designs

Proposed by [Spe10, VS10]

Variables = 1s in matrix with four 1s per row/column + extra 1

Each row wants majority true; each column wants majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Subset Cardinality Formulas / Zero-One Designs

Proposed by [Spe10, VS10]

Variables = 1s in matrix with four 1s per row/column + extra 1

Each row wants majority true; each column wants majority false

$$
\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \color{red}{\mathbf{1}} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \color{red}{\mathbf{1}} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \color{red}{\mathbf{1}} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \color{red}{\mathbf{1}}
\end{pmatrix}
$$

$$
\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \quad \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}
$$

Subset Cardinality Formulas / Zero-One Designs

Proposed by [Spe10, VS10]

Variables $=$ 1s in matrix with four 1s per row/column $+$ extra 1

Each row wants majority true; each column wants majority false

$$\begin{pmatrix}
\mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 & 0 \\
0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} & 0 \\
0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & \mathbf{1} \\
\mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 \\
0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} \\
\mathbf{1} & 0 & \mathbf{1} & 0 & 0 & 0 & \mathbf{1} & \mathbf{1} & 0 & 0 & \mathbf{1}
\end{pmatrix}$$

$$\begin{aligned}
& (x_{1,1} \lor x_{1,2} \lor x_{1,4}) \\
\land\ & (x_{1,1} \lor x_{1,2} \lor x_{1,8}) \\
\land\ & (x_{1,1} \lor x_{1,4} \lor x_{1,8}) \\
\land\ & (x_{1,2} \lor x_{1,4} \lor x_{1,8}) \\
& \vdots \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{10,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{8,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{4,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11}) \\
\land\ & (\overline{x}_{8,11} \lor \overline{x}_{10,11} \lor \overline{x}_{11,11})
\end{aligned}$$

Hard for expanding (well spread-out) matrices [MN14]

but easy for regular patterns like the one above (even for DPLL)

Ordering Principle Formulas

"Every finite ordered set $\{e_1, \ldots, e_n\}$ has minimal element"

Variables $x_{i,j} =$ "$e_i < e_j$"

$\overline{x}_{i,j} \lor \overline{x}_{j,i}$	anti-symmetry; not both $e_i < e_j$ and $e_j < e_i$
$\overline{x}_{i,j} \lor \overline{x}_{j,k} \lor x_{i,k}$	transitivity; $e_i < e_j$ and $e_j < e_k$ implies $e_i < e_k$
$\bigvee_{1 \leq i \leq n,\, i \neq j} x_{i,j}$	e_j is not a minimal element

Can also add "total order" axioms

$x_{i,j} \lor x_{j,i}$	totality; either $e_i < e_j$ or $e_j < e_i$

Ordering Principle Formulas

"Every finite ordered set $\{e_1, \ldots, e_n\}$ has minimal element"

Variables $x_{i,j} =$ "$e_i < e_j$"

$\overline{x}_{i,j} \vee \overline{x}_{j,i}$	anti-symmetry; not both $e_i < e_j$ and $e_j < e_i$
$\overline{x}_{i,j} \vee \overline{x}_{j,k} \vee x_{i,k}$	transitivity; $e_i < e_j$ and $e_j < e_k$ implies $e_i < e_k$
$\bigvee_{1 \leq i \leq n,\, i \neq j} x_{i,j}$	e_j is not a minimal element

Can also add "total order" axioms

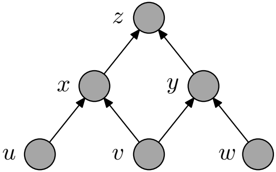
$x_{i,j} \vee x_{j,i}$	totality; either $e_i < e_j$ or $e_j < e_i$

Conjectured hard [Kri85] but refutable in length $\mathcal{O}(N)$ [Stå96]
Requires resolution width $\Omega(\sqrt[3]{N})$ converted to k-CNF [BG01]
(Or use asymmetric width measure in [Kul99])

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

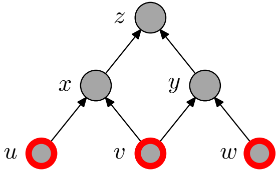


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \land (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \land (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \land (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

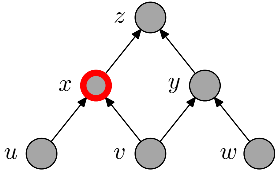


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

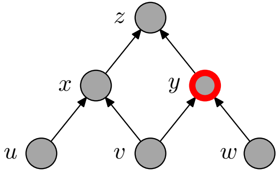


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

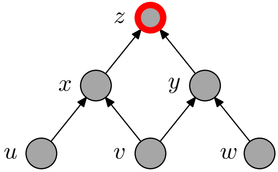


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

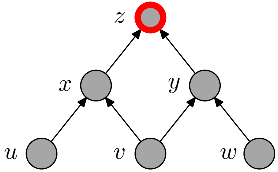


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \land (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \land (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \land (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$

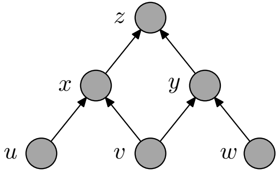


- sources are true
- truth propagates upwards
- but sink is false

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \land (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \land (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \land (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$



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

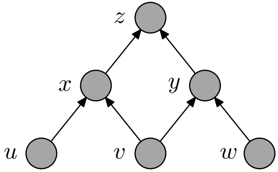
Write in CNF; e.g., $(x_1 \oplus x_2) \rightarrow (y_1 \oplus y_2)$ becomes

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

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$



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

Write in CNF; e.g., $(x_1 \oplus x_2) \rightarrow (y_1 \oplus y_2)$ becomes

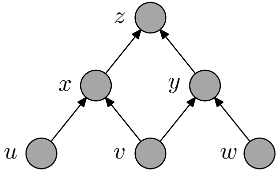
$$\begin{aligned} &(x_1 \vee \overline{x}_2 \vee y_1 \vee y_2) \wedge (x_1 \vee \overline{x}_2 \vee \overline{y}_1 \vee \overline{y}_2) \\ \wedge\ &(\overline{x}_1 \vee x_2 \vee y_1 \vee y_2) \wedge (\overline{x}_1 \vee x_2 \vee \overline{y}_1 \vee \overline{y}_2) \end{aligned}$$

Pebble game trade-offs $\Rightarrow$ resolution size-space trade-offs [BN08, BN11]

Pebbling Formulas

Encode so-called pebble games on DAGs [BW01]

1. $u_1 \oplus u_2$
2. $v_1 \oplus v_2$
3. $w_1 \oplus w_2$
4. $(u_1 \oplus u_2) \wedge (v_1 \oplus v_2) \rightarrow (x_1 \oplus x_2)$
5. $(v_1 \oplus v_2) \wedge (w_1 \oplus w_2) \rightarrow (y_1 \oplus y_2)$
6. $(x_1 \oplus x_2) \wedge (y_1 \oplus y_2) \rightarrow (z_1 \oplus z_2)$
7. $\neg(z_1 \oplus z_2)$



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

Write in CNF; e.g., $(x_1 \oplus x_2) \rightarrow (y_1 \oplus y_2)$ becomes

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

Pebble game trade-offs $\Rightarrow$ resolution size-space trade-offs [BN08, BN11]
Works for other functions than $\oplus$ (we use NEQ_3, but harder to illustrate)

Instrumented CDCL Solver

To run experiments, add "knobs" to Glucose [AS09, Glu] to analyse:

- restart policy
- branching
- clause database management
- clause learning

Instrumented CDCL Solver

To run experiments, add "knobs" to Glucose [AS09, Glu] to analyse:

- restart policy
- branching
- clause database management
- clause learning

Though strictly speaking not part of basic CDCL, we also study effects of:

- preprocessing (Glucose standard on/off; so far always on)
- random shuffling of instances (but doesn't seem to matter)

Instrumented CDCL Solver

To run experiments, add “knobs” to Glucose [AS09, Glu] to analyse:

- restart policy
- branching
- clause database management
- clause learning

Though strictly speaking not part of basic CDCL, we also study effects of:

- preprocessing (Glucose standard on/off; so far always on)
- random shuffling of instances (but doesn't seem to matter)

Yields huge number of potential combinations

- Not all combinations make sense, but many do
- Test also settings where “convential wisdom” knows answer
- Several settings still remain to test — marked with $*$ in what follows

CDCL Parameters (1/2)

Restart policy

- No restarts
- LBD-style restarts (Glucose)
- Luby restarts (with different multiplicative factors)*

Variable selection

- Fixed order (chosen to be good)
- VSIDS (with decay factors 0.99^*, 0.95, 0.80, 0.65^*)

Phase saving

- Random phase
- Phase fixed to all false at start of execution*
- Phase fixed randomly at start of execution*
- Standard phase saving

CDCL Parameters (2/2)

Clause erasure

- No clause deletion (keep all learned clauses)
- "Classic" MiniSat-style removal ($\Theta(n)$ clauses after n conflicts)*
- Glucose-style removal ($\Theta(\sqrt{n})$ clauses after n conflicts)
- New, more aggressive MiniSat ($O(n^{0.24})$ clauses after n conflicts)

Clause assessment

- Keep clauses with a good (high) VSIDS score à la MiniSat
- Keep clauses with a good (low) LBD score à la Glucose

Clause learning

- DPLL-style search with minimal amount of clause learning*
- Standard 1UIP clause learning

Some Preliminary Conclusions (1/2)

Importance of restarts

- Sometimes very frequent restarts very important
- Crucial in [AFT11, PD11] for CDCL to simulate resolution efficiently
- Also seems to matter in practice for some formulas which are hard for subsystems of resolution such as regular resolution (stone formulas)

Some Preliminary Conclusions (1/2)

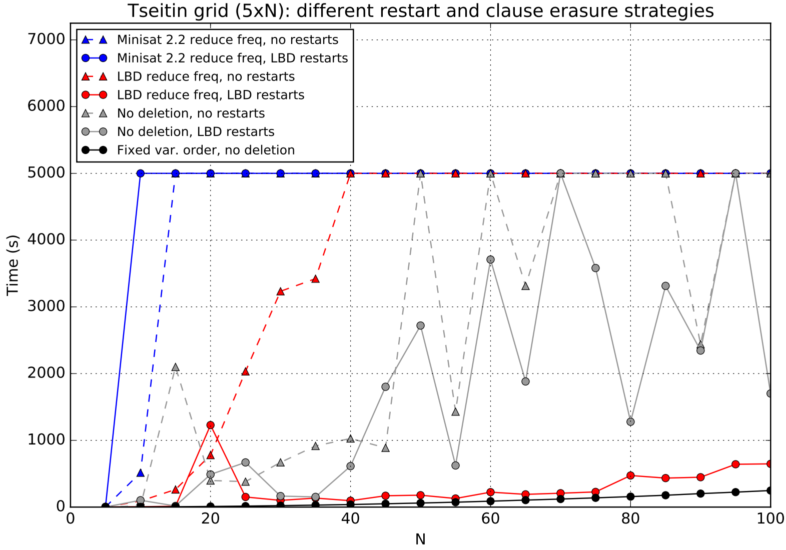
Importance of restarts

- Sometimes very frequent restarts very important
- Crucial in [AFT11, PD11] for CDCL to simulate resolution efficiently
- Also seems to matter in practice for some formulas which are hard for subsystems of resolution such as regular resolution (stone formulas)

Clause erasure

- Theory says very aggressive clause removal could hurt badly
- Seem to see this on scaled-down versions of time-space trade-off formulas in [BBI12, BNT13] (Tseitin formulas)
- Even no erasure at all can be competitive for these formulas for frequent enough restarts

Plot 1: Tseitin Formulas on Grids



Some Preliminary Conclusions (2/2)

Clause assessment

- Can LBD (literal block distance) heuristic compensate for aggressive erasures by identifying important clauses to keep? Maybe. . .
- But LBD can backfire for too aggressive removal — old glue clauses clog up the clause database(?)

Some Preliminary Conclusions (2/2)

Clause assessment

- Can LBD (literal block distance) heuristic compensate for aggressive erasures by identifying important clauses to keep? Maybe...
- But LBD can backfire for too aggressive removal — old glue clauses clog up the clause database(?)

Variable branching

- Phase saving only helps together with frequent restarts
- Sometimes small variations in VSIDS decay factor (rate of forgetting) absolutely crucial (ordering principle)
- Does slow decay bring solver closer to tree-like resolution???

Some Preliminary Conclusions (2/2)

Clause assessment

- Can LBD (literal block distance) heuristic compensate for aggressive erasures by identifying important clauses to keep? Maybe...
- But LBD can backfire for too aggressive removal — old glue clauses clog up the clause database(?)

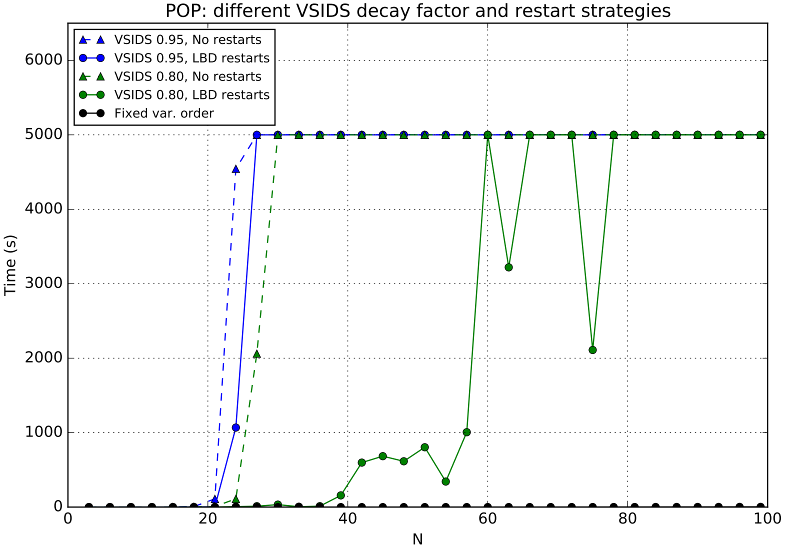
Variable branching

- Phase saving only helps together with frequent restarts
- Sometimes small variations in VSIDS decay factor (rate of forgetting) absolutely crucial (ordering principle)
- Does slow decay bring solver closer to tree-like resolution???

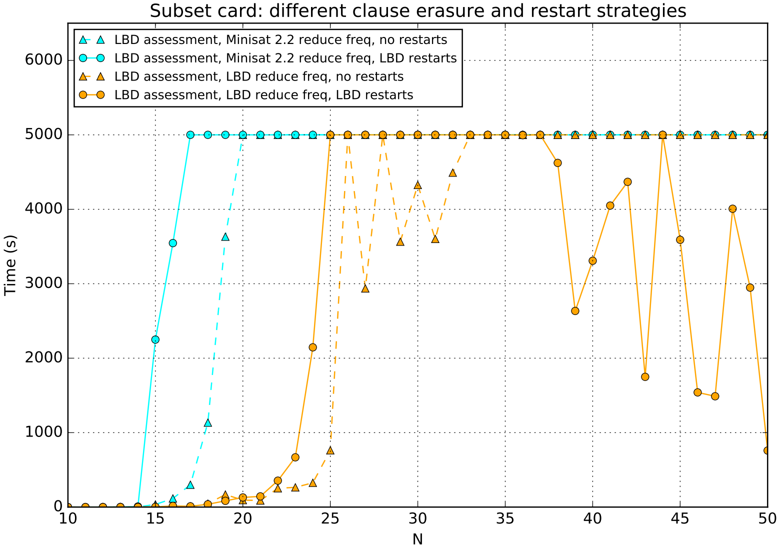
CDCL vs. resolution

- Sometimes CDCL fails miserably on easy formulas (Tseitin, even colouring) — VSIDS just goes dead wrong
- Sometimes strange easy-hard-easy patterns (zero-one designs)

Plot 2: Ordering Principle Formulas



Plot 3: Zero-One Designs



Concluding Remarks

- Report on a work still very much in progress

Concluding Remarks

- Report on a work still very much in progress
- Hoping to stimulate discussion and generate feed-back (already happened with PoS reviews)

Concluding Remarks

- Report on a work still very much in progress
- Hoping to stimulate discussion and generate feed-back
  (already happened with PoS reviews)
- Main novelty: experiments on scalable & easy theoretical benchmarks

Concluding Remarks

- Report on a work still very much in progress
- Hoping to stimulate discussion and generate feed-back
  (already happened with PoS reviews)
- Main novelty: experiments on scalable & easy theoretical benchmarks
- Main purpose not to solve more SAT competion benchmarks, but to gain better understanding of CDCL

Concluding Remarks

- Report on a work still very much in progress
- Hoping to stimulate discussion and generate feed-back (already happened with PoS reviews)
- Main novelty: experiments on scalable & easy theoretical benchmarks
- Main purpose not to solve more SAT competion benchmarks, but to gain better understanding of CDCL
- Presented only some preliminary findings — we are optimistic that further interesting conclusions will follow

Concluding Remarks

- Report on a work still very much in progress
- Hoping to stimulate discussion and generate feed-back (already happened with PoS reviews)
- Main novelty: experiments on scalable & easy theoretical benchmarks
- Main purpose not to solve more SAT competion benchmarks, but to gain better understanding of CDCL
- Presented only some preliminary findings — we are optimistic that further interesting conclusions will follow

Thank you for your attention!

References I

[ABRW02] Michael Alekhnovich, Eli Ben-Sasson, Alexander A. Razborov, and Avi Wigderson. Space complexity in propositional calculus. *SIAM Journal on Computing*, 31(4):1184–1211, 2002. Preliminary version in *STOC '00*.

[AD08] Albert Atserias and Víctor Dalmau. A combinatorial characterization of resolution width. *Journal of Computer and System Sciences*, 74(3):323–334, May 2008. Preliminary version in *CCC '03*.

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

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

[ALN16] Albert Atserias, Massimo Lauria, and Jakob Nordström. Narrow proofs may be maximally long. *ACM Transactions on Computational Logic*, 17:19:1–19:30, May 2016. Preliminary version in *CCC '14*.

References II

[AMO13] Albert Atserias, Moritz Müller, and Sergi Oliva. Lower bounds for DNF-refutations of a relativized weak pigeonhole principle. In *Proceedings of the 28th Annual IEEE Conference on Computational Complexity (CCC '13)*, pages 109–120, June 2013.

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

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

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

[BG03] Eli Ben-Sasson and Nicola Galesi. Space complexity of random formulae in resolution. *Random Structures and Algorithms*, 23(1):92–109, August 2003. Preliminary version in *CCC '01*.

References III

[BN08] Eli Ben-Sasson and Jakob Nordström. Short proofs may be spacious: An optimal separation of space and length in resolution. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, pages 709–718, October 2008.

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

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

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

[CNF] CNFgen formula generator and tools. `https://github.com/MassimoLauria/cnfgen`.

[CS88] Vašek Chvátal and Endre Szemerédi. Many hard examples for resolution. *Journal of the ACM*, 35(4):759–768, October 1988.

References IV

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

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

[ET01] Juan Luis Esteban and Jacobo Torán. Space bounds for resolution. *Information and Computation*, 171(1):84–97, 2001. Preliminary versions of these results appeared in *STACS '99* and *CSL '99*.

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

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

[Kri85] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, August 1985.

References V

[KSM11] Hadi Katebi, Karem A. Sakallah, and João P. Marques-Silva. Empirical study of the anatomy of modern SAT solvers. In *Proceedings of the 14th International Conference on Theory and Applications of Satisfiability Testing (SAT '11)*, volume 6695 of *Lecture Notes in Computer Science*, pages 343–356. Springer, June 2011.

[Kul99] Oliver Kullmann. Investigating a general hierarchy of polynomially decidable classes of CNF's based on short tree-like resolution proofs. Technical Report TR99-041, Electronic Colloquium on Computational Complexity (ECCC), 1999.

[LENV16] Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals. CNFgen: a generator of crafted CNF formulas. Manuscript in preparation, 2016.

[Mar06] Klas Markström. Locality and hard SAT-instances. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):221–227, 2006.

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

References VI

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

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

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

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

[Spe10] Ivor Spence. sgen1: A generator of small but difficult satisfiability benchmarks. *Journal of Experimental Algorithmics*, 15:1.2:1.1–1.2:1.15, March 2010.

[Stå96] Gunnar Stålmarck. Short resolution proofs for a sequence of tricky formulas. *Acta Informatica*, 33(3):277–280, May 1996.

References VII

[Tse68] Grigori Tseitin. On the complexity of derivation in propositional calculus. In A. O. Silenko, editor, *Structures in Constructive Mathematics and mathematical Logic, Part II*, pages 115–125. Consultants Bureau, New York-London, 1968.

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

[VS10] Allen Van Gelder and Ivor Spence. Zero-one designs produce small hard SAT instances. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 388–397. Springer, July 2010.