Seeking Practical CDCL Insights from Theoretical SAT Benchmarks

to appear in IJCAI 2018

Jan Elffers, Jesús Giráldez Cru, **Stephan Gocht**,
Jakob Nordström and Laurent Simon

29.05.2018

The SAT Problem

- Literal a: Boolean 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

Has F satisfying assignment?

The Power of so called CDCL SAT Solvers

2017 SAT Competition [BHJ17]

- largest solved benchmark (g2-T96.1.1.cnf)
  - 8 905 808 variables
  - 32 322 587 clauses
  - verifiable UNSAT in 4126.12s
- smallest unsolved (mp1-bsat222-777.cnf)
  - 222 variables
  - 777 clauses
  - timelimit 5000s

Explanation?

Understanding Performance

Problem instance determines:

- solver performance
- which algorithms / heuristics are important / good

Solvers essentially do resolution
$\Rightarrow$ well understood through proof complexity

- scalable UNSAT problems
- extremal w.r.t. certain property
  $\Rightarrow$ lower bound on runtime
- expect different behaviour

Our Project

Goal:

- understand which / when settings are important

Our approach for reaching this goal:

- crafted benchmarks[1], using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

[1] generated using CNFGen [LENV17]

Related Work

- instrumentation [LM02, KSM11]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

Related Work

- instrumentation [LM02, KSM11]
- decision heuristics [BF15]
- restart schemes [Hua07]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

Related Work

- instrumentation [LM02, KSM11]
- decision heuristics [BF15]
- restart schemes [Hua07]
- structural restricted benchmarks [PJ09]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

Related Work

- instrumentation [LM02, KSM11]
- decision heuristics [BF15]
- restart schemes [Hua07]
- structural restricted benchmarks [PJ09]
- random k-SAT [CA96, SLM92]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

Related Work

- instrumentation [LM02, KSM11]
- decision heuristics [BF15]
- restart schemes [Hua07]
- structural restricted benchmarks [PJ09]
- random k-SAT [CA96, SLM92]
- analysing and evaluating theory formula [MN14]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

Related Work

- instrumentation [LM02, KSM11]
- decision heuristics [BF15]
- restart schemes [Hua07]
- structural restricted benchmarks [PJ09]
- random k-SAT [CA96, SLM92]
- analysing and evaluating theory formula [MN14]
- resolution space on theory formula [JMNŽ12]

Our approach:

- crafted benchmarks, using knowledge from proof complexity
- benchmarks are
  - scalable
  - easy
  - extremal (or close to)
- instrument solver to switch between algorithms / heuristics

The CDCL Algorithm [DP60, DLL62, MS99, MMZ+01, ...]

Used Implementations: MiniSat [ES04], Glucose [AS09]

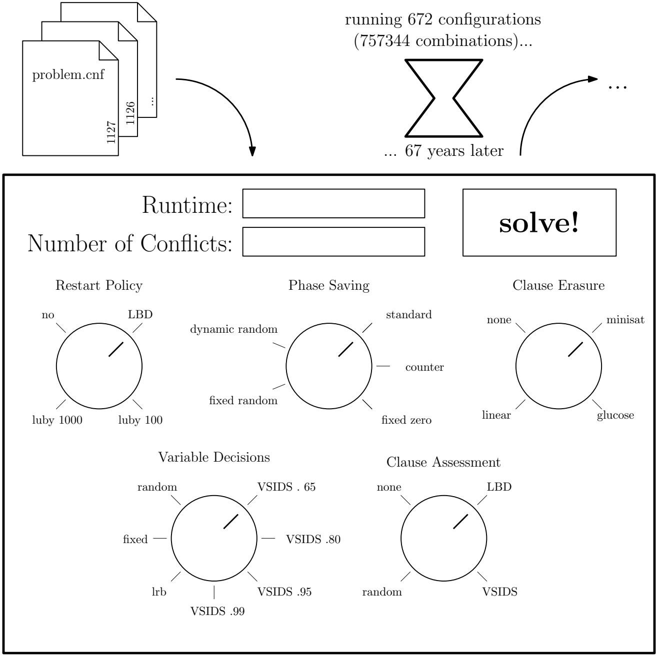
```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit (fact) propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT

    return SAT
```

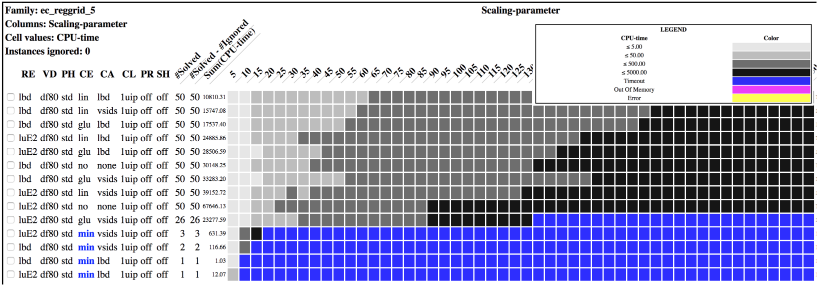
The CDCL Algorithm [DP60, DLL62, MS99, MMZ+01, ...]

Used Implementations: MiniSat [ES04], Glucose [AS09]

```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit (fact) propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT
            k ← amount of clause erasure
            if k > 0 then
                remove k clauses with bad clause assessment
            if time for restart then
                undo all decisions
    return SAT
```


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

Heatmaps



- row: setting
- column: scaled instances
- colour: runtime

Analysing PAR-Score

PAR-X-score: runtime if solved, otherwise $X \cdot$ timelimit
($X = 2$ used)

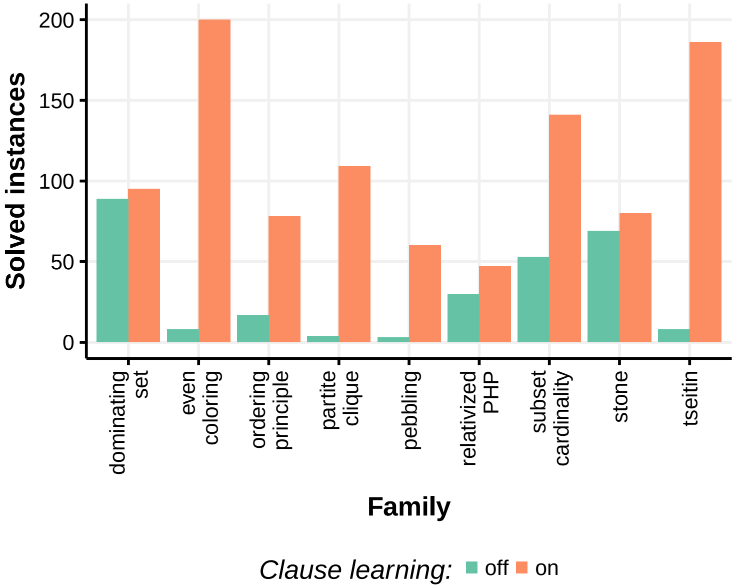
Analyse:

- fix some “knobs”
- compute expected score
  (average of settings containing fixed “knobs”)
- compare to global average, but:
  - always some difference
  - choose random subset of settings
    $\Rightarrow$ yields standard deviation
    (used to “value” expected score)

The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT
            k ← amount of clause erasure
            if k > 0 then
                remove k clauses with bad clause assessment
            if time for restart then
                undo all decisions
    return SAT
```

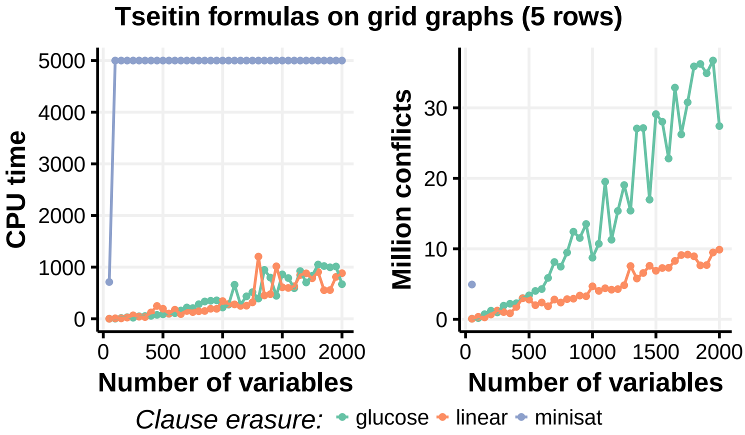
Clause Learning, Going Beyond Treelike Resolution



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT
            k ← amount of clause erasure
            if k > 0 then
                remove k clauses with bad clause assessment
            if time for restart then
                undo all decisions
    return SAT
```

DB Size on Theoretical Time-Space Trade-Off Formulas

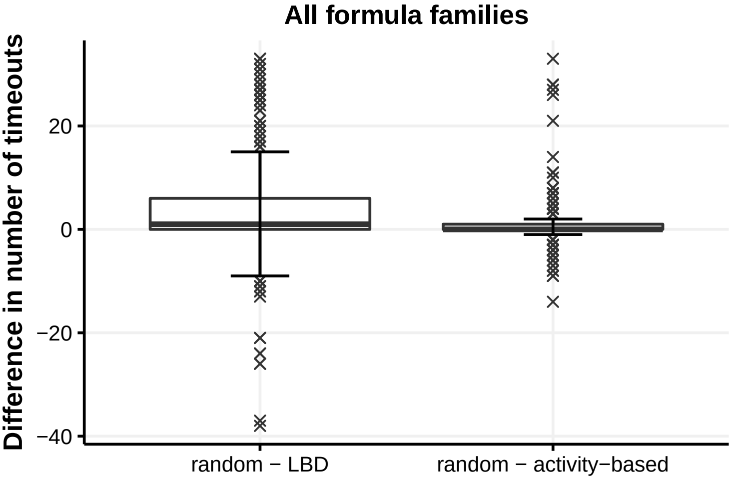


database size: minisat $<$ glucose $<$ linear

The CDCL Algorithm

procedure SOLVE(F)
    while $v \leftarrow$ next variable decision **do**
        assign v to chosen phase
        do unit propagation
        if conflict **then**
            add clause learned from conflict
            if decision to be undone **then** undo bad decisions
            else return UNSAT
            $k \leftarrow$ amount of clause erasure
            if $k > 0$ **then**
                remove k clauses with bad clause assessment
            if time for restart **then**
                undo all decisions
    return SAT

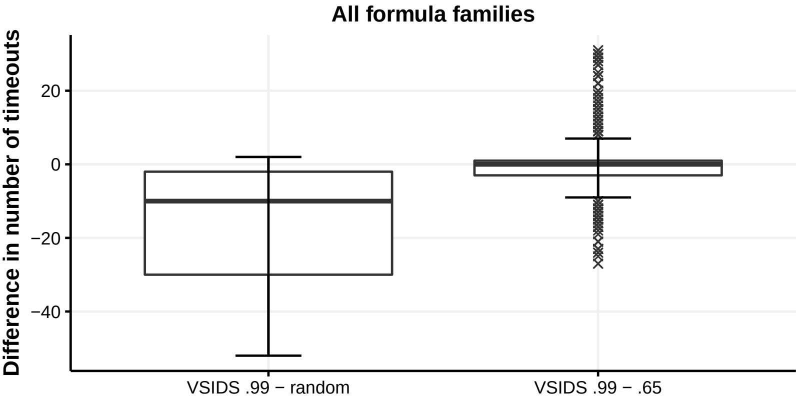
Clause Assessment



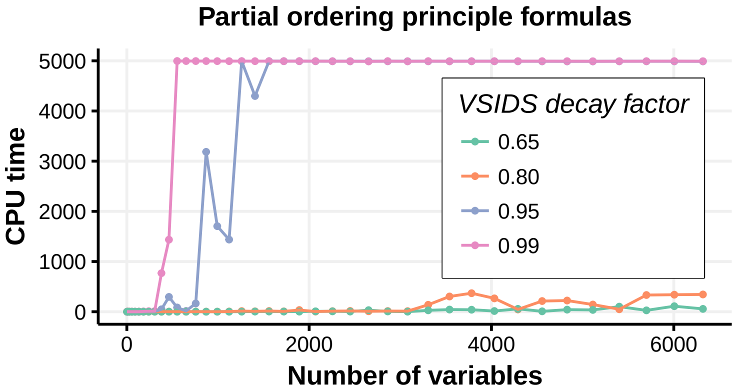
The CDCL Algorithm

procedure SOLVE(F)
    while $v \leftarrow$ next variable decision **do**
        assign v to chosen phase
        do unit propagation
        if conflict **then**
            add clause learned from conflict
            if decision to be undone **then** undo bad decisions
            else return UNSAT
            $k \leftarrow$ amount of clause erasure
            if $k > 0$ **then**
                remove k clauses with bad clause assessment
            if time for restart **then**
                undo all decisions
    return SAT

Variable Decision



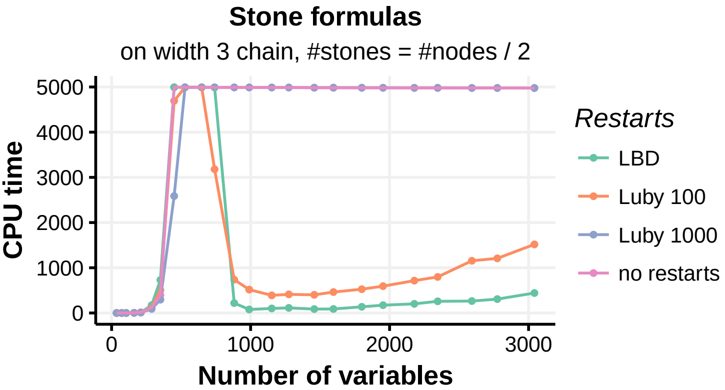
Variable Decision



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT
            k ← amount of clause erasure
            if k > 0 then
                remove k clauses with bad clause assessment
            if time for restart then
                undo all decisions
    return SAT
```

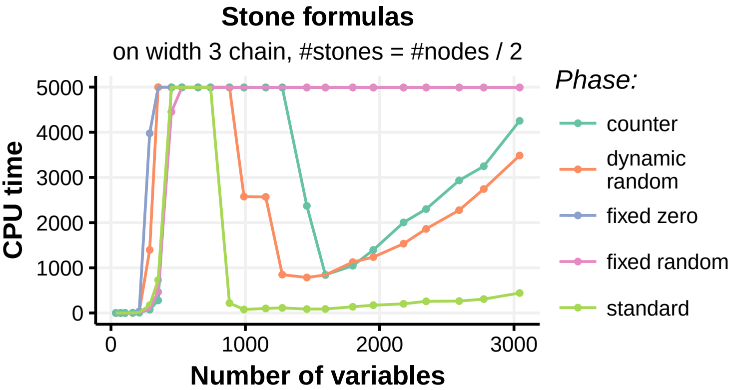
Restarts for Unrestricted Resolution



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        assign v to chosen phase
        do unit propagation
        if conflict then
            add clause learned from conflict
            if decision to be undone then undo bad decisions
            else return UNSAT
            k ← amount of clause erasure
            if k > 0 then
                remove k clauses with bad clause assessment
            if time for restart then
                undo all decisions
    return SAT
```

Phase Saving



Conclusions

- clause learning is important
  (if you need to go beyond treelike resolution)
- choose the *right* database size
  (required space vs. overhead)
- restarts help to harness the full power of resolution
  (if necessary)
- VSIDS is good for variable decisions
  (but can go badly wrong)

Conclusions

- clause learning is important
  (if you need to go beyond treelike resolution)
- choose the *right* database size
  (required space vs. overhead)
- restarts help to harness the full power of resolution
  (if necessary)
- VSIDS is good for variable decisions
  (but can go badly wrong)

Thank you for your attention!

References I

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.

Armin Biere and Andreas Fröhlich.
Evaluating CDCL variable scoring schemes.
In *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT '15)*, volume 9340 of *Lecture Notes in Computer Science*, pages 405–422. Springer, September 2015.

Tomáš Balyo, Marijn JH Heule, and Matti Jäarvisalo.
Proceedings of sat competition 2017: Solver and benchmark descriptions.
2017.

James M. Crawford and Larry D. Auton.
Experimental results on the crossover point in random 3-SAT.
Artificial Intelligence, 81(1-2):31–57, March 1996.
Preliminary version in *AAAI '93*.

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

References II

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

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

Jinbo Huang.
The effect of restarts on the efficiency of clause learning.
In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pages 2318–2323, January 2007.

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.

References III

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.

Massimo Lauria, Jan Elffers, Jakob Nordström, and Marc Vinyals.
CNFgen: A generator of crafted benchmarks.
In *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT '17)*, volume 10491 of *Lecture Notes in Computer Science*, pages 464–473. Springer, August 2017.

Inês Lynce and João P. Marques-Silva.
Building state-of-the-art SAT solvers.
In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI '02)*, pages 166–170. IOS Press, May 2002.

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 IV

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.

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*.

Justyna Petke and Peter Jeavons.
Tractable benchmarks for constraint programming.
Technical Report RR-09-07, Oxford University Computing Laboratory, 2009.
Available at `https://www.cs.ox.ac.uk/files/2366/RR-09-07.pdf`.

Bart Selman, Hector J. Levesque, and David G. Mitchell.
A new method for solving hard satisfiability problems.
In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI '92)*, pages 440–446, July 1992.