Seeking Practical CDCL Insights from Theoretical SAT Benchmarks

to appear at IJCAI 2018

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

27.08.2018

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

Does F have satisfying assignment?

About SAT

- NP-complete [Coo71]
  $\Rightarrow$ believed to be very hard
- *conflict driven clause learning* (CDCL) solvers
  [MS99, BS97, MMZ$^+$01, ...]
  very efficient $\Rightarrow$ makes practical problems tractable
- based on many sophisticated heuristics
- not very well understood:
  - Which heuristics are important and why?
  - How do heuristics interact?

Problems with SAT competition benchmarks

- very heterogeneous benchmarks
- poorly understood properties
  $\Rightarrow$ isolated data points
  $\Rightarrow$ inconclusive results
- limited selection of benchmarks
  $\Rightarrow$ solvers might already be over-fitted

Alternative: Proof Complexity

- method of reasoning used by CDCL: resolution
- resolution is extremely well studied in theory

Pros:

- lots of theoretically well understood SAT problems
- scalable $\Rightarrow$ "same problem" in different sizes
- sometimes multiple versions of same size

Cons:

- only considers existence of proofs
- results are asymptotic
  (Required parameters reasonably small?)
- crafted benchmarks are... crafted

Our Approach

- choose / tune theory benchmarks, such that:
  - cover different extremal properties
    $\Rightarrow$ "stress test" heuristics
  - easy in theory (should be tractable)
    $\Rightarrow$ measure quality of proof search
  - scalable
    $\Rightarrow$ measure asymptotic performance
- instrument solver to switch between heuristics
- essentially run full cross product of heuristics

Our Approach

- choose / tune theory benchmarks, such that:
  - cover different extremal properties
    $\Rightarrow$ "stress test" heuristics
  - easy in theory (should be tractable)
    $\Rightarrow$ measure quality of proof search
  - scalable
    $\Rightarrow$ measure asymptotic performance
- instrument solver to switch between heuristics
- essentially run full cross product of heuristics

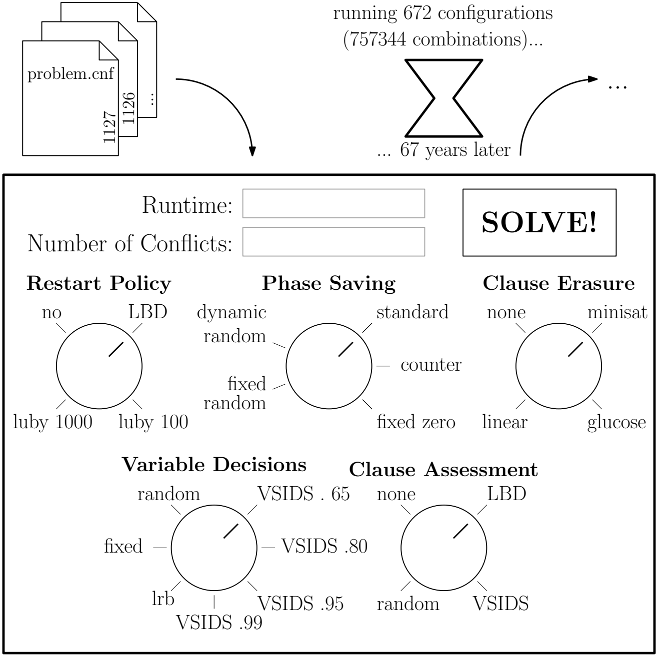
Related Work:

- comparing heuristics (not to the extend we do)
  [BF15, Hua07, LM02, KSM11]
- crafted benchmarks (just one family of benchmarks)
  [PJ09, CA96, SLM92, MN14, JMNŽ12]

The CDCL Algorithm [MS99, BS97, MMZ+01, ...]

Instrumented Solver: Glucose [AS09] / MiniSat [ES04]

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst 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

The Experiments

- 27 (sub)families of formulas
- 1127 instances
- 672 solver configurations
- over 500'000 hours (67 years)
- measure running time, #decisions, #conflicts, ...
- huge amount of data ...

Mining Data

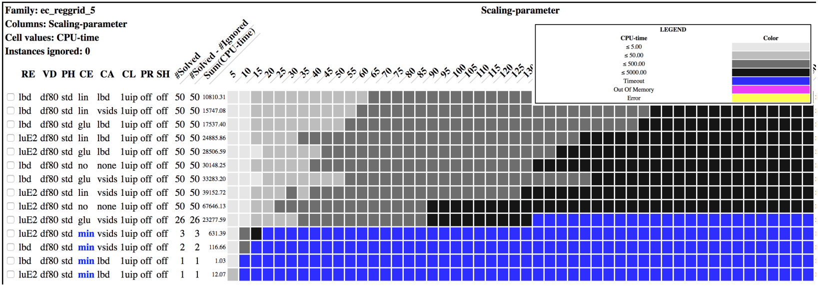
Research Question:

- Which heuristics are most important for specific family?
- Improvement due to one heuristics independent from others?

Challenge:

- How to even get an overview of data?
- How to avoid invalid conclusions?
- solvers deterministic — essentially no random noise
- no standard tool to compare deterministic algorithms

Heatmaps



- row: setting
- column: scaled instances
- colour: running time

Available online:

`https://www.csc.kth.se/~jakobn/CDCL-insights`

Analysing PAR-Score

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

Analysing:

- use resampling approach
  (compare x to y randomly drawn from same data as x)
- used to mine the data
- no concrete claims about significance (p-value)

The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

Theory: Clause Learning and Tree-Like Resolution

- search in CDCL solvers crucially guided by conflicts
- clause learning influences:
  - variable decisions
  - restart policy
  - memory management
  - and more...
- is storing learned clauses also crucial?

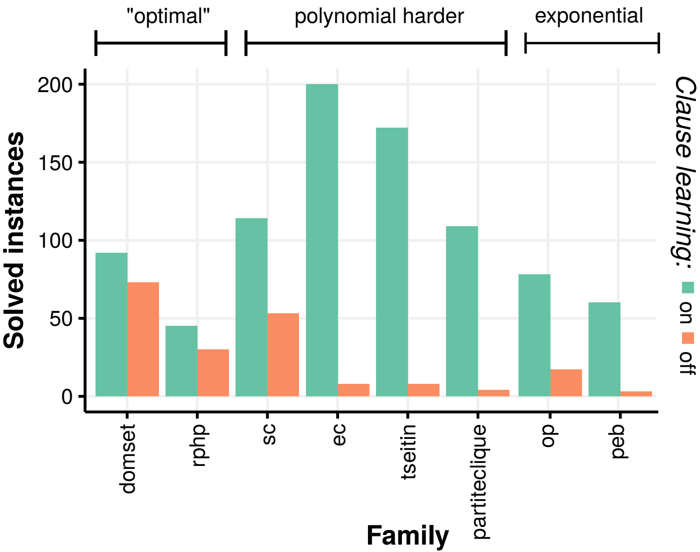
Theory

- no learning $\Rightarrow$ tree-like resolution (DPLL)

How to check in practice?

- instances that separate tree-like and general resolution
- instances where tree-like is “optimal”

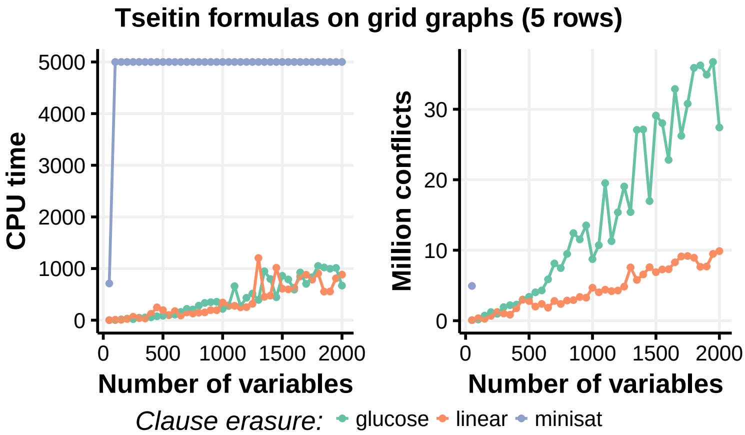
Empirical: Clause Learning and Tree-Like Resolution



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

Memory Management and Theoretical Time-Space Trade-Offs

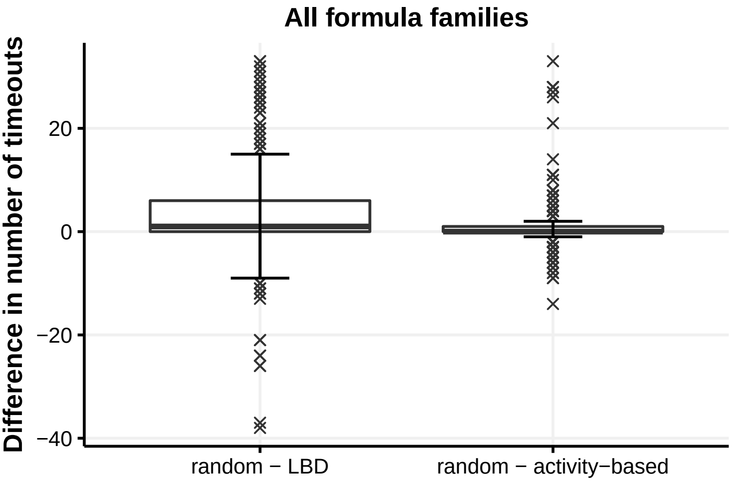


database size: minisat ($\sim N^{0.25}$) $<$ glucose ($\sim N^{0.5}$) $<$ linear ($\sim N$)
($N =$ number of conflicts)

The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

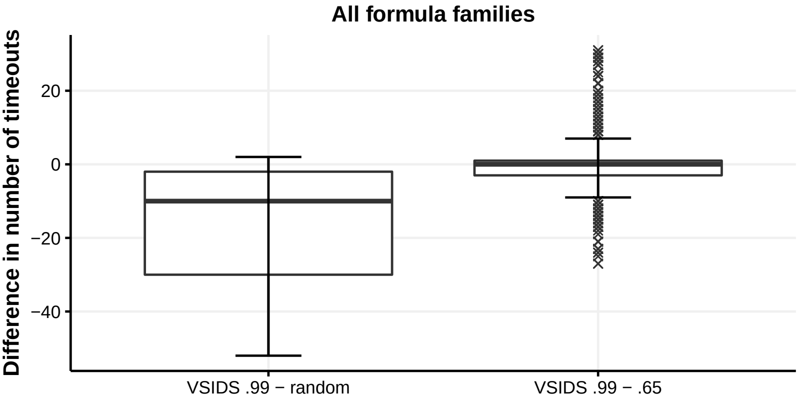
Clause Assessment



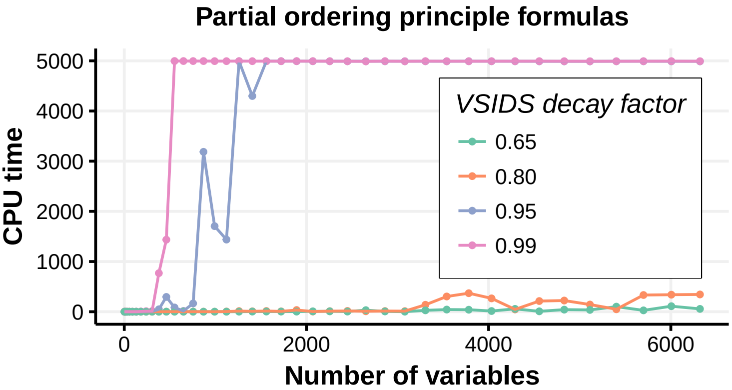
The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

Variable Decision



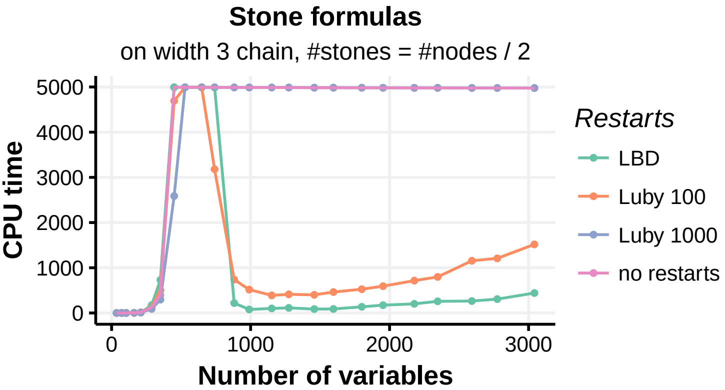
Variable Decision



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

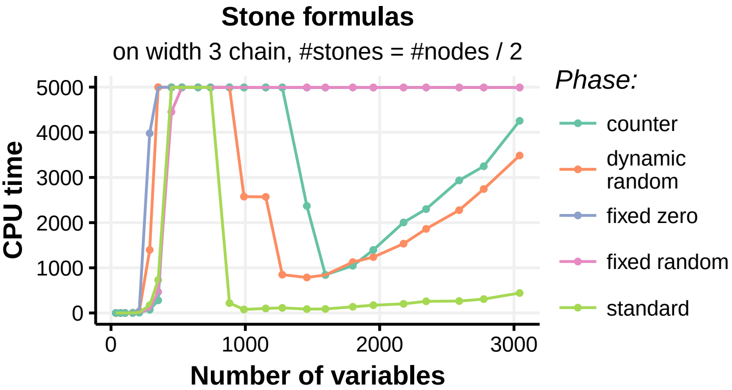
Restarts for Unrestricted Resolution



The CDCL Algorithm

```
procedure SOLVE(F)
    while v ← next variable decision do
        decide on v with chosen phase
        do unit (fact) propagation
        if conflict (there is falsified clause) then
            if no decided variable then return UNSAT
            learn clause from conflict
            backjump (undo bad decisions)
        if time to prune clause database then
            k ← difference to new database size
            remove k clauses with worst clause assessment
        if time for restart then
            undo all decisions
    return SAT
```

Phase Saving



Overview of Results

- storing learned clauses is important
  (if you need to go beyond treelike resolution)
- size of database can be critical
  (trade-off between propagation speed and proof quality (?))
- frequent restarts $\Rightarrow$ stronger proof search (?)
  (for formulas where full power of resolution needed)
- assessing quality of learned clauses challenging
  (LBD mostly works well; activity-based $\sim$ random)
- variable decisions absolutely crucial
  (random terrible, VSIDS good but can go badly wrong)

Conclusion

Can get practical CDCL insights from theoretical SAT benchmarks!

- confirmation of conventional wisdom
  (nice to see evidence)
- used benchmarks highlight strengths and weaknesses of heuristics
- sometimes raises intriguing open questions:
  - Restarts: only frequency important or timing as well?
  - More learned clauses always better for proof search?
  - VSIDS decay factor sometimes crucial — how to choose?
  - ...

Conclusion

Can get practical CDCL insights from theoretical SAT benchmarks!

- confirmation of conventional wisdom
  (nice to see evidence)
- used benchmarks highlight strengths and weaknesses of heuristics
- sometimes raises intriguing open questions:
  - Restarts: only frequency important or timing as well?
  - More learned clauses always better for proof search?
  - VSIDS decay factor sometimes crucial — how to choose?
  - ...

Thank you for your attention!

References I

[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.

[BF15] 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.

[BS97] Roberto J. Bayardo Jr. and Robert Schrag.
Using CSP look-back techniques to solve real-world SAT instances.
In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI '97)*, pages 203–208, July 1997.

[CA96] 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*.

References II

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

[ES04] 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.

[Hua07] 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.

[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.

References III

[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.

[LM02] 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.

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

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

References IV

[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*.

[PJ09] 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`.

[SLM92] 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.