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 $\bar{x}$ (or $\neg x$)

▶ Clause $C = a_1 \lor \cdots \lor a_k$: disjunction of literals
  (Consider as sets, so no repetitions and order irrelevant)

▶ CNF formula $F = C_1 \land \cdots \land C_m$: conjunction of clauses

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
⇒ well understood through proof complexity

▶ scalable \textit{UNSAT} problems
▶ extremal w.r.t. certain property
  ⇒ 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]

1: **procedure** `SOLVE(F)`
2: **while** \( v \leftarrow \) next variable decision **do**
3: assign \( v \) to chosen phase
4: do unit (fact) propagation
5: **if** conflict **then**
6: add clause learned from conflict
7: **if** decision to be undone **then** undo bad decisions
8: **else** return UNSAT

14: return SAT
The CDCL Algorithm

[DP60, DLL62, MS99, MMZ⁺01, …]

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

1: **procedure** `SOLVE(F)`
2: while $v \leftarrow$ next variable decision do
3:    assign $v$ to chosen phase
4:    do unit (fact) propagation
5:    **if** conflict **then**
6:       add clause learned from conflict
7:    **if** decision to be undone **then** undo bad decisions
8:    **else** return UNSAT
9: $k \leftarrow$ amount of clause erasure
10: **if** $k > 0$ **then**
11:    remove $k$ clauses with bad clause assessment
12: **if** time for restart **then**
13:    undo all decisions
14: return SAT
Stephan Gocht
CDCL on Theory Benchmarks
8/24

running 672 configurations
(757344 combinations)...

... 67 years later

Runtime: ___________________________
Number of Conflicts: _______________
solve!

Restart Policy
Phase Saving
Clause Erasure

no
LBD
luby 100
luby 1000

standard
counter
none
minisat

dynamic random
fixed random
fixed zero
linear

Variable Decisions
Clause Assessment

random
fixed
lrb

VSIDS .99

VSIDS .65

VSIDS .80

VSIDS .95

VSIDS
Heatmaps

- **row:** setting
- **column:** scaled instances
- **colour:** runtime
Analysing PAR-Score

**PAR-X-score**: runtime if solved, otherwise $X \cdot \text{timelimit}$

($X = 2$ 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

1: procedure solve(F)
2: while ν ← next variable decision do
3: assign ν to chosen phase
4: do unit propagation
5: if conflict then
6: add clause learned from conflict
7: if decision to be undone then undo bad decisions
8: else return UNSAT
9: k ← amount of clause erasure
10: if k > 0 then
11: remove k clauses with bad clause assessment
12: if time for restart then
13: undo all decisions
14: return SAT
Clause Learning, Going Beyond Treelike Resolution

Clause learning:

- off
- on
The CDCL Algorithm

1: procedure \textsc{solve}(F)
2: \hspace{1em} while \( v \leftarrow \text{next variable decision} \) do
3: \hspace{2em} assign \( v \) to chosen phase
4: \hspace{2em} do unit propagation
5: \hspace{2em} \textbf{if} conflict \textbf{then}
6: \hspace{3em} add clause learned from conflict
7: \hspace{2em} \textbf{if} decision to be undone \textbf{then} undo bad decisions
8: \hspace{2em} \textbf{else} return UNSAT
9: \hspace{2em} \( k \leftarrow \text{amount of clause erasure} \)
10: \hspace{2em} \textbf{if} \( k > 0 \) \textbf{then}
11: \hspace{3em} remove \( k \) clauses with bad clause assessment
12: \hspace{2em} \textbf{if} time for restart \textbf{then}
13: \hspace{3em} undo all decisions
14: \hspace{2em} return SAT
DB Size on Theoretical Time-Space Trade-Off Formulas

Tseitin formulas on grid graphs (5 rows)

Clause erasure: glucose < linear < minisat

database size: minisat < glucose < linear
The CDCL Algorithm

1: procedure SOLVE\((F)\)
2: while \(v \leftarrow\) next variable decision do
3: assign \(v\) to chosen phase
4: do unit propagation
5: if conflict then
6: add clause learned from conflict
7: if decision to be undone then undo bad decisions
8: else return UNSAT
9: \(k \leftarrow\) amount of clause erasure
10: if \(k > 0\) then
11: remove \(k\) clauses with bad clause assessment
12: if time for restart then
13: undo all decisions
14: return SAT
Clause Assessment

All formula families

Difference in number of timeouts

random – LBD

random – activity-based
The CDCL Algorithm

1: procedure solve$(F)$
2: while $\nu \leftarrow$ next variable decision do
3: assign $\nu$ to chosen phase
4: do unit propagation
5: if conflict then
6: add clause learned from conflict
7: if decision to be undone then undo bad decisions
8: else return UNSAT
9: $k \leftarrow$ amount of clause erasure
10: if $k > 0$ then
11: remove $k$ clauses with bad clause assessment
12: if time for restart then
13: undo all decisions
14: return SAT
Variable Decision

All formula families

Difference in number of timeouts

VSIDS .99 – random

VSIDS .99 – .65

Stephan Gocht

CDCL on Theory Benchmarks
Variable Decision

Partial ordering principle formulas

VSIDS decay factor
- 0.65
- 0.80
- 0.95
- 0.99

CPU time

Number of variables
The CDCL Algorithm

1: procedure $\text{SOLVE}(F)$
2: while $v \leftarrow \text{next variable decision}$ do
3: assign $v$ to chosen phase
4: do unit propagation
5: if conflict then
6: add clause learned from conflict
7: if decision to be undone then undo bad decisions
8: else return UNSAT
9: $k \leftarrow \text{amount of clause erasure}$
10: if $k > 0$ then
11: remove $k$ clauses with bad clause assessment
12: if time for restart then
13: undo all decisions
14: return SAT

Stephan Gocht CDCL on Theory Benchmarks 20/ 24
Restarts for Unrestricted Resolution

Stone formulas
on width 3 chain, #stones = #nodes / 2

CPU time
Number of variables

Restarts
- LBD
- Luby 100
- Luby 1000
- no restarts

Stephan Gocht
CDCL on Theory Benchmarks
The CDCL Algorithm

1:  procedure \texttt{SOLVE}(F)
2:  \hspace{1em} \textbf{while} $v \leftarrow$ next variable decision \textbf{do}
3:    assign $v$ to chosen phase
4:    do unit propagation
5:    \hspace{1em} \textbf{if} conflict \textbf{then}
6:        add clause learned from conflict
7:        \hspace{1em} \textbf{if} decision to be undone \textbf{then} undo bad decisions
8:        \hspace{1em} \textbf{else} return UNSAT
9:    \hspace{1em} $k \leftarrow$ amount of clause erasure
10:    \hspace{1em} \textbf{if} $k > 0$ \textbf{then}
11:        remove $k$ clauses with bad clause assessment
12:    \hspace{1em} \textbf{if} time for restart \textbf{then}
13:        undo all decisions
14:    \hspace{1em} return SAT
Phase Saving

Stone formulas
on width 3 chain, #stones = #nodes / 2

Phase:
- counter
- dynamic
- random
- fixed zero
- fixed random
- standard

CPU time vs. Number of variables graph:
- counter
- dynamic
- random
- fixed zero
- fixed random
- standard

Stephan Gocht
CDCL on Theory Benchmarks 23/24
Conclusions

- clause learning is important
  (if you need to go beyond treelike resolution)
- choose the \textit{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


References II


References IV


