Decision levels are stable: towards better SAT heuristics

LPAR 23, January 2021 (invited talk)

Robert Nieuwenhuis

(Joint work with Adrià Lozano, Albert Oliveras, Enric Rodríguez-Carbonell)

Barcelogic.com - Computer Science Department
Founder and CEO - Professor
BarcelonaTech (UPC)

Thanks for inviting me!

Made in de snow in El Perelló, Catalonia!

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
    - (cloud-based) solvers, from SAT to CP, SMT and ILP
    - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD
- Introducing *stickiness*

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD
- Introducing *stickiness*
- Experiments: stickiness depends on the problem, not on a run, strategy, or encoding!

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD
- Introducing *stickiness*
- Experiments: stickiness depends on the problem, not on a run, strategy, or encoding!
- How quickly does stickiness stabilize in a run?

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
    - (cloud-based) solvers, from SAT to CP, SMT and ILP
    - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD
- Introducing *stickiness*
- Experiments: stickiness depends on the problem, not on a run, strategy, or encoding!
- How quickly does stickiness stabilize in a run?
- How stickiness explains and improves LBD

Outline of this talk

- Barcelogic and UPC BarcelonaTech: solver development
- Aims:
  - (cloud-based) solvers, from SAT to CP, SMT and ILP
  - beat commercial MIP solvers on the harder combinatorial problems
- The basis of all this: CDCL SAT solvers. Why do they work so well?
- Crucial heuristics in CDCL: which learned clauses to keep (and share)?
- *Literal Block Distance* (LBD) and *glue*-based heuristics
- VIGs and other attempts to understand LBD
- Introducing *stickiness*
- Experiments: stickiness depends on the problem, not on a run, strategy, or encoding!
- How quickly does stickiness stabilize in a run?
- How stickiness explains and improves LBD
- Exploiting stickiness: insights gained and consequences

About Barcelogic

- Spin-off company from BarcelonaTech (UPC).
- Core Team: PhDs in Math, CS.
- Customers in Logistics, HRM, Routing, Sports, ...

Football:

FIFA, Leagues in Spain, Italy, Mexico, Holland, Portugal, Denmark ...

Other sports:

Basketball, Cricket, Paralympics...

FIFA WC draw, Moscow 2017: Gary Lineker using Barcelogic Draw Assistant

Interest in solvers from SAT to SMT, ILP, our IntSat method

Joint project between Barcelogic and BarcelonaTech.
At Barcelogic, sports' customers pay the bills for this ongoing R&D.

Barcelogic does Combinatorial Optimization with Logic-based modelling, and emphasis on problems that are Combinatorial rather than numerical.
This is Hard! Good heuristics (AI) are essential for feasibility and optimization.

The future (we think): highly parallel cloud-based solvers for SAT, ILP, Constraint Programming, SMT...

(see LPAR in red on this slide).

From SAT to Integer Linear Programming (ILP)

	0-1 solutions		$\mathbb{Z}$ solutions	
	feasibility	optimizing	feasibility	optimizing
clauses	SAT			
cardinality constr.				
linear constraints				ILP

ILP: Find solution : $\{x_1 \dots x_n\} \to \mathbb{Z}$ to:

Minimize: $c_1 \; x_1 + \ldots + c_n \; x_n$ (or maximize)

Subject To:

$$
\begin{array}{rcl}
c_{11}\, x_1 + \ldots + c_{1n}\, x_n & \geq & c_{10} \\
\ldots & & \ldots \\
c_{m1}\, x_1 + \ldots + c_{mn}\, x_n & \geq & c_{m0}
\end{array}
$$

(or with $\leq, =, <, >$)

where c_i in $\mathbb{Z}$.

(Card.: $x_1 + \ldots + x_n \geq c_0$)

SAT: particular case of ILP with 0-1 vars and where constraints are clauses:

$$x \vee \overline{y} \vee z \quad \equiv \quad x + (1 - y) + z \geq 1 \quad \equiv \quad x - y + z \geq 0$$

And From SAT to SAT Modulo Theories (SMT)

SMT:

Deciding satisfiability of a SAT formula with atoms over a background theory T

Example 1: T is Equality with Uninterpreted Functions (EUF):

3 clauses: $f(g(a)) \neq f(c) \vee g(a) = d, \qquad g(a) = c, \qquad c \neq d$

Example 2: several (how many?) combined theories:

2 clauses: $A = write(B, i+1, x), \qquad read(A, j+3) = y \vee f(i-1) \neq f(j+1)$

Typical verification examples, where SMT is method of choice.

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Example. Four clauses:

$\overline{1} \vee 2, \;\; \overline{3} \vee 4, \;\; \overline{5} \vee \overline{6}, \;\; 6 \vee \overline{5} \vee \overline{2}$

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution: **Example. Four clauses:**

	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$		

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$		*CONFLICT*!

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Backtrack)

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$	$\Rightarrow$	(Backtrack)
1 2 3 4 $\overline{5}$	$\overline{1}\lor 2,\ \overline{3}\lor 4,\ \overline{5}\lor\overline{6},\ 6\lor\overline{5}\lor\overline{2}$		

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Decide)
$1\ 2\ 3\ 4\ 5$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(UnitPropagate)
$1\ 2\ 3\ 4\ 5\ \overline{6}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$	$\Rightarrow$	(Backtrack)
$1\ 2\ 3\ 4\ \overline{5}$	$\overline{1} \vee 2,\ \overline{3} \vee 4,\ \overline{5} \vee \overline{6},\ 6 \vee \overline{5} \vee \overline{2}$		solution found!

The Basis of all this: CDCL-based SAT Solvers

After decades of industry $$$:

high-performance, complete, general-purpose, push-button.

CDCL = Conflict-Driven Clause-Learning backtracking/backjumping algorithm

Candidate Solution:	**Example. Four clauses:**		
	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Decide)
1 2 3 4 5	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(UnitPropagate)
1 2 3 4 5 $\overline{6}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$	$\Rightarrow$	(Backtrack)
1 2 3 4 $\overline{5}$	$\overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2}$		solution found!

Can do much better! Next: Backjump instead of Backtrack...

Backtrack vs. Backjump

Same example. Remember: Backtrack gave $1\ 2\ 3\ 4\ \overline{5}$.

But: decision level $3\ 4$ is irrelevant for the conflict $6{\vee}\overline{5}{\vee}\overline{2}$:

$$
\begin{array}{lllll}
\emptyset & \| & \overline{1}{\vee}2,\ \overline{3}{\vee}4,\ \overline{5}{\vee}\overline{6},\ 6{\vee}\overline{5}{\vee}\overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots & \vdots & \vdots & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \| & \overline{1}{\vee}2,\ \overline{3}{\vee}4,\ \overline{5}{\vee}\overline{6},\ 6{\vee}\overline{5}{\vee}\overline{2} & \Rightarrow & (\text{Backjump})
\end{array}
$$

Backtrack vs. Backjump

Same example. Remember: Backtrack gave $1\ 2\ 3\ 4\ \overline{5}$.

But: decision level $3\ 4$ is irrelevant for the conflict $6 \vee \overline{5} \vee \overline{2}$:

$$
\begin{array}{lll}
\emptyset & \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad (\text{Decide}) \\
\vdots & \quad\vdots \qquad\qquad \vdots & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad (\text{Backjump}) \\
1\ 2\ \overline{5} & \overline{1} \vee 2,\ \ \overline{3} \vee 4,\ \ \overline{5} \vee \overline{6},\ \ 6 \vee \overline{5} \vee \overline{2} & \Rightarrow \quad \ldots
\end{array}
$$

Backtrack vs. Backjump

Same example. Remember: Backtrack gave $1\ 2\ 3\ 4\ \overline{5}$.

But: decision level $3\ 4$ is irrelevant for the conflict $6\vee\overline{5}\vee\overline{2}$:

$$\begin{array}{llll}
\emptyset & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Decide}) \\
\vdots & \vdots & & \\
1\ 2\ 3\ 4\ 5\ \overline{6} & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & (\text{Backjump}) \\
1\ 2\ \overline{5} & \overline{1}\vee 2,\ \overline{3}\vee 4,\ \overline{5}\vee\overline{6},\ 6\vee\overline{5}\vee\overline{2} & \Rightarrow & \ldots
\end{array}$$

Backjump =

1. Conflict Analysis: "Find" a backjump clause $C \vee l$ (here, $\overline{2}\vee\overline{5}$)
   - that is a logical consequence of the clause set
   - that reveals a unit propagation of l at an earlier decision level d (i.e., where its part C is false)
2. Return to decision level d and do that propagation.

Conflict Analysis: find backjump clause

Example. Consider the stack: $\ldots 6 \ldots \overline{7} \ldots 9$ and clauses:

$\overline{6} \vee 7 \vee \overline{9} \vee \overline{8}$, $8 \vee 7 \vee \overline{5}$, $\overline{6} \vee 8 \vee 4$, $\overline{4} \vee \overline{1}$, $\overline{4} \vee 5 \vee 2$, $5 \vee 7 \vee \overline{3}$, $1 \vee \overline{2} \vee 3$

UnitPropagate gives $\ldots 6 \ldots \overline{7} \ldots 9\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$. Conflict w/ $1 \vee \overline{2} \vee 3$!

C.An. = do resolutions with reason clauses backwards from conflict:

$$\frac{5 \vee 7 \vee \overline{3} \qquad 1 \vee \overline{2} \vee 3}{5 \vee 7 \vee 1 \vee \overline{2}}$$

$$\frac{\overline{4} \vee 5 \vee 2 \qquad 5 \vee 7 \vee 1 \vee \overline{2}}{\overline{4} \vee 5 \vee 7 \vee 1}$$

$$\frac{\overline{4} \vee \overline{1} \qquad \overline{4} \vee 5 \vee 7 \vee 1}{5 \vee 7 \vee \overline{4}}$$

$$\frac{\overline{6} \vee 8 \vee 4 \qquad 5 \vee 7 \vee \overline{4}}{\overline{6} \vee 8 \vee 7 \vee 5}$$

$$\frac{8 \vee 7 \vee \overline{5} \qquad \overline{6} \vee 8 \vee 7 \vee 5}{8 \vee 7 \vee \overline{6}}$$

until get clause with only 1 literal of last decision level. "1-UIP"

Can use this backjump clause $8 \vee 7 \vee \overline{6}$ to Backjump to $\ldots 6 \ldots \overline{7}\ 8$.

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002].   Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver (“forgetting” the theory T)

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver ("forgetting" the theory T)

   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002].   Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver (“forgetting” the theory T)

   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver ("forgetting" the theory T)
   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$
   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent
2. Send $\{\ \ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ to SAT solver

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver ("forgetting" the theory T)

   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent
2. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ to SAT solver

   SAT solver returns model $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver (“forgetting” the theory T)

   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent
2. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ to SAT solver

   SAT solver returns model $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$ is T-inconsistent

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver ("forgetting" the theory T)
   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$
   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent
2. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ to SAT solver
   SAT solver returns model $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$
   Theory solver says $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$ is T-inconsistent
3. Send $\{\overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4,\ \ \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\ \}$ to SAT solver

SAT as a basis for SMT. Lazy SMT

Aka Lemmas on demand [dMR,2002]. Same EUF example as before:

$$\underbrace{f(g(a)) \neq f(c)}_{\overline{1}} \vee \underbrace{g(a) = d}_{2}, \qquad \underbrace{g(a) = c}_{3}, \qquad \underbrace{c \neq d}_{\overline{4}}$$

1. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4}\ \}$ to SAT solver ("forgetting" the theory T)

   SAT solver returns model $[\ \overline{1},\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ \overline{1},\ 3,\ \overline{4}\ ]$ is T-inconsistent
2. Send $\{\ \overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4\ \}$ to SAT solver

   SAT solver returns model $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$

   Theory solver says $[\ 1,\ 2,\ 3,\ \overline{4}\ ]$ is T-inconsistent
3. Send $\{\overline{1} \vee 2,\ \ 3,\ \ \overline{4},\ \ 1 \vee \overline{3} \vee 4,\ \ \overline{1} \vee \overline{2} \vee \overline{3} \vee 4\ \}$ to SAT solver

   SAT solver says UNSAT

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models
- Check T-consistency of partial assignment while being built

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause
- Given a T-inconsistent assignment M, find an explanation (a small T-inconsistent subset of M) and add it as a clause

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause
- Given a T-inconsistent assignment M, find an explanation (a small T-inconsistent subset of M) and add it as a clause

- Upon a T-inconsistency, add clause and restart

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Improved Lazy approach

Since state-of-the-art SAT solvers are all DPLL-based...

- Check T-consistency only of full propositional models
- Check T-consistency of partial assignment while being built

- Given a T-inconsistent assignment M, add $\neg M$ as a clause
- Given a T-inconsistent assignment M, find an explanation (a small T-inconsistent subset of M) and add it as a clause

- Upon a T-inconsistency, add clause and restart
- Upon a T-inconsistency, do conflict analysis of the explanation and Backjump

Note: in CP, this is called Lazy Clause Generation (Stuckey et al).

Current standard: our DPLL(T) approach to SMT (JACM'06)

DPLL(T) = DPLL(X) engine + *T*-Solvers

- Modular and flexible: can plug in any *T*-Solvers into the DPLL(X) engine.
- *T*-Solvers specialized and fast in Theory Propagation:
  - Propagate literals that are theory consequences
  - more pruning in improved lazy SMT
  - *T*-Solver also guides search, instead of only validating it
  - fully exploited in conflict analysis (non-trivial)
- DPLL(X) engine is essentially a SAT solver.

Back to SAT. Why is CDCL really that good?

Three key ingredients (I think):

Back to SAT. Why is CDCL really that good?

Three key ingredients (I think):

1. Learn at each conflict backjump clause as a lemma (“nogood”):
    - makes UnitPropagate more powerful
    - prevents EXP repeated work in future similar conflicts

Back to SAT. Why is CDCL really that good?

Three key ingredients (I think):

1. Learn at each conflict backjump clause as a lemma (“nogood”):
   - makes UnitPropagate more powerful
   - prevents EXP repeated work in future similar conflicts

2. Decide on variables with many occurrences in Recent conflicts:
   - Dynamic activity-based variabe selection heuristics
   - idea: work off, one by one, clusters of tightly-related vars
     (try CDCL on two independent instances together...)

Back to SAT. Why is CDCL really that good?

Three key ingredients (I think):

1. Learn at each conflict backjump clause as a lemma ("nogood"):
   - makes UnitPropagate more powerful
   - prevents EXP repeated work in future similar conflicts
2. Decide on variables with many occurrences in Recent conflicts:
   - Dynamic activity-based variabe selection heuristics
   - idea: work off, one by one, clusters of tightly-related vars (try CDCL on two independent instances together...)
3. Forget from time to time less important lemmas:
   - crucial to keep UnitPropagate fast and memory affordable
   - THIS WORK:
     - Which lemmas ARE important?
     - Which ones to share in parallel solvers?

Literal Block Distance (LBD) and *glue*-based heuristics

Literal Block Distance (LBD) of a lemma: the number of decision levels (dl's) involved when it was generated.

Same example: UnitPropagate gave $\ldots 6 \ldots \overline{7} \ldots 9\ \overline{8}\ \overline{5}\ 4\ \overline{1}\ 2\ \overline{3}$. Conflict!

Learned lemma $8 \vee 7 \vee \overline{6}$ used to Backjump to $\ldots 6 \ldots \overline{7}\ 8$.

If 6 and $\overline{7}$ are at different decision levels, then this lemma has LBD = 3.

Glue clauses: lemmas with LBD = 2. They "glue" together two dl's.

Glucose solvers (Audemard, Simon)

Most solvers never delete glue clauses. Some re-compute LBD values from time to time.

Idea: the fewer dl's, the more likely the lemma becomes useful again....

if..... decision levels are "stable"!

Introducing Stickiness in a CDCL run R

Idea: two variables are "sticky" means "they are frequently at the same dl":

Formally: $stick_R(x, y)$ is the (conditional) probability that,
if we pick a dl with x or y, that also the other one is at that dl:

$$stick_R(x, y) \quad = \quad \frac{n_R(x, y)}{n_R(x) + n_R(y) - n_R(x, y)}$$

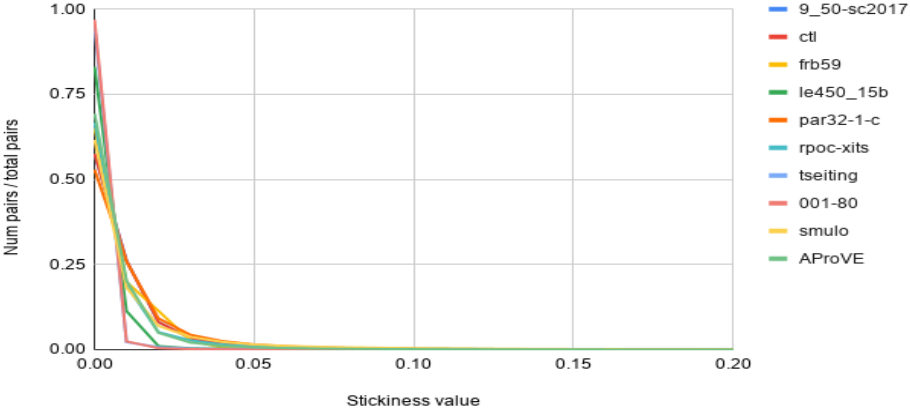
where $n_R(x)$ and $n_R(x, y)$ denote # dl's containing x or both.

Notes: $stick_R(x, y) \in [0 \dots 1]$ and:

- $stick_R(x, y) = 1$ if x and y are always together when assigned
- $stick_R(x, y)$ is demanding: it quickly drops.
  E.g., if $n_R(x) = n_R(y)$ of which 80% together, then it is only $80/(100 + 100 - 80) = 0.66$.
- $stick_R(x, y) \approx 0$ for almost all pairs (x, y).

Most pairs have very low stickiness:

All experiments with 10 industrial instances (randomly selected from SAT Race) and four state-of-the-art solvers (Cadical1,2,3 and Glucose).



Average over the four solvers. X-axis cut at 0.2: too few have stickiness > 0.2.

How similar are two runs R and R' in stickiness?

$$Sim(R,R') \quad = \quad \frac{\displaystyle\sum_{\{x,y\}\subseteq\mathcal{X}} \quad min(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}{\displaystyle\sum_{\{x,y\}\subseteq\mathcal{X}} \quad max(\ stick_R(x,y),\ stick_{R'}(x,y)\ )}$$

Notes: $Sim(R,R') \in [0 \ldots 1]$ and:

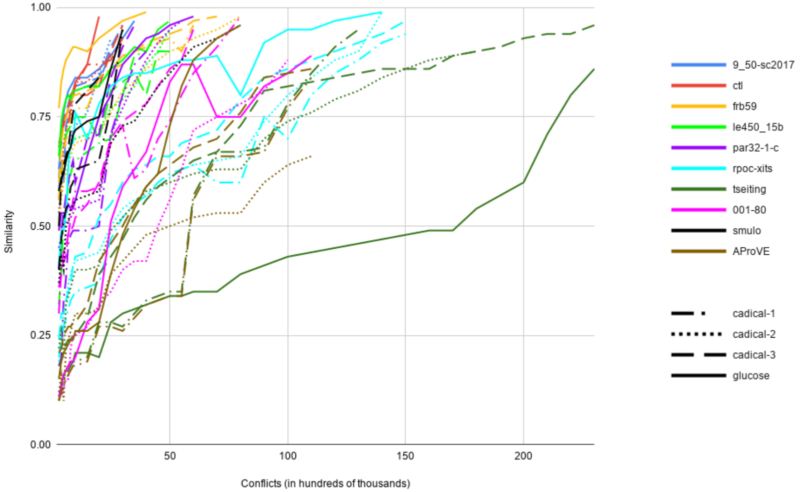
- it is closer to 1 the more pairs (x,y) are similarly sticky in R and in R', and it weights high-stickiness similarities more than low ones
- it is also demanding: it quickly drops.
  E.g., if $stick_R(x,y) = 0.3$ and $stick_{R'}(x,y) = 0.1$ for many pairs, this pushes $Sim(R,R')$ to 0.33
- it is close to 0 for two identical runs except random permutation of vars
- Remarkable key result: it is still > 0.7 and up to 0.9 between runs with different random seeds and even between different solvers! (see below)

Unrelated runs indeed have low similarity

	cad1	cad2	cad3	glu
9-50-sc2017	0.04	0.09	0.02	0.01
ctl	0.03	0.07	0.10	0.04
frb59	0.01	0.07	0.07	0.08
le450-15b	0.05	0.02	0.05	0.07
par32-1-c	0.09	0.01	0.07	0.04
rpoc-xits	0.04	0.08	0.03	0.10
tseiting	0.07	0.07	0.08	0.09
001-80	0.03	0.04	0.06	0.10
smulo	0.06	0.03	0.09	0.01
AProVE	0.05	0.03	0.10	0.01

Similarity of two runs of same solver, seed and instance, but with randomly permuted variables. Always ≤ 0.1.

How does stickiness evolve along a run of a solver?



40 runs (10 instances x 4 solvers); Similarity w.r.t. final stickiness.

Most runs very quickly stabilize (but not all).

Comparing different solvers, still similar stickiness

	cad1-2	cad1-3	cad1-glu	cad2-3	cad2-glu	cad3-glu
9-50-sc2017	0.76	0.79	0.65	0.68	0.67	0.58
ctl	0.75	0.71	0.55	0.68	0.54	0.56
frb59	0.82	0.86	0.50	0.79	0.51	0.50
le450-15b	0.75	0.76	0.53	0.64	0.52	0.55
par32-1-c	0.77	0.60	0.55	0.64	0.66	0.53
rpoc-xits	0.68	0.70	0.53	0.63	0.54	0.53
tseiting	0.63	0.50	0.48	0.52	0.51	0.56
001-80	0.63	0.50	0.51	0.57	0.55	0.53
smulo	0.65	0.50	0.48	0.53	0.51	0.56
AProVE	0.68	0.50	0.52	0.54	0.51	0.61

Same solver, different random seeds: even more similar

	cad1	cad2	cad3	glu
9-50-sc2017	0.90	0.93	0.93	0.90
ctl	0.78	0.71	0.70	0.70
frb59	0.87	0.82	0.80	0.87
le450-15b	0.87	0.88	0.94	0.89
par32-1-c	0.92	0.86	0.85	0.90
rpoc-xits	0.77	0.75	0.76	0.79
tseiting	0.60	0.61	0.64	0.69
001-80	0.62	0.59	0.59	0.68
smulo	0.60	0.60	0.68	0.68
AProVE	0.68	0.66	0.73	0.69

Even under different cardinality constraint encodings!

Real-world Barcelogic professional sport scheduling problem with many cardinality constraints.

Comparing:

R: direct encoding

$>$ 4 million clauses, 30K variables

R': our sophisticated encoding

600 K clauses, adds 17K auxiliary vars to the 30K

For each solver, compare stickiness similarity among the 30K original vars:

cad1	cad2	cad3	glu
0.65	0.74	0.68	0.73

Remarkable, especially since solvers decide on auxiliary variables too!

VIGs: static attempts to understand LBD

In a Variable Incidence Graph (VIG) of a CNF, nodes are variables, and (weighted) edges indicate their occurrences in the same clause.

VIGs from industrial instances can be split into communities with few inter-community edges.

But... Different encodings give completely different VIGs!

We do not support at all (see our paper) certain results in the literature relating

a) the LBD of a clause

with

b) the number of VIG communities it involves.

We found zero relationship between VIG's edges' weights and stickiness.

How can our new insights help improving solvers?

Remember: LBD-based cleanups are standard in state-of-the-art SAT.

Laurent Simon (2019 Dagstuhl seminar discussion):

"LBD values are probably meaningful only during the same run".

But, even in the same run, and even if periodically updated, LBD values are still imprecise snapshots.

Idea: Stable LBD of a clause = its no. of subsets of highly-sticky literals

Idea: Share in parallel solvers: units, binaries,... and low Stable LBD clauses?

Idea: Preprocess or in-process computing low Stable LBD clauses.

(for all three, experimental assessment is ongoing)

Conclusion

Stickiness (+ community structure derived from it) captures useful and stable instance properties.

Highly unlikely to be possible from static info from the CNF alone.

We believe this this will be crucial for future SAT, SMT, Pseudo-Boolean, ILP solvers as well as LCG-based Constraint programming, and especially for (cloud-based) versions thereof.

Thank you!