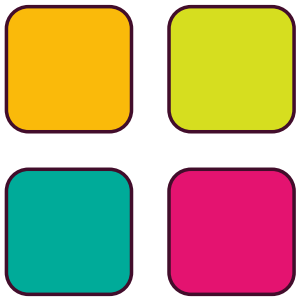


CDCL and parallel SAT

Gilles Audemard

SAT Summer School - Manchester - 2018

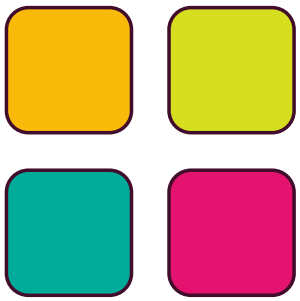
Thanks to N. Szczepanski and L. Simon

Summary

- Introduction
- CDCL solvers
- Parallel solvers
- SAT incremental

`https://tinyurl.com/ybr6mf3v`

`https://github.com/audemard/minicdcl`



Introduction

SAT Solving in Practice

- SAT is a success story in computer science
- Many practical applications

Source J. Marques-Silva

You already know that...

Why studing CDCL solvers

- The logic behind SAT is simple
- A CDCL solver is not too difficult to implement (a version with 500 loc is available with the presentation)
- Many open source solvers are available : Minisat is the most famous one
- You can easily modify them and try new techniques/ideas
- You need to think carefully with data-structures (many variables/clauses)

But.. many promising ideas are finally ineffective

Playing with SAT solvers is addictive

From DP to CDCL

1960 Davis and Putnam algorithm : based on resolution rule , forget one variable after the other

Resolution rule

$$(x \vee y \vee \neg z) \otimes_x (\neg x \vee y \vee t) = y \vee \neg z \vee t$$

From DP to CDCL

1960 Davis and Putnam algorithm : based on resolution rule , forget one variable after the other

Resolution rule

$$(x \vee y \vee \neg z) \otimes_x (\neg x \vee y \vee t) = y \vee \neg z \vee t$$

$$x_1 \vee x_4$$
$$\neg x_1 \vee x_4 \vee x_{14}$$
$$\neg x_1 \vee \neg x_3 \vee \neg x_8$$
$$x_1 \vee x_8 \vee x_{12}$$
$$x_1 \vee x_5 \vee \neg x_9$$
$$x_2 \vee x_{11}$$
$$\neg x_3 \vee \neg x_7 \vee x_{13}$$
$$\neg x_3 \vee \neg x_7 \vee \neg x_{13} \vee x_9$$
$$x_8 \vee \neg x_7 \vee \neg x_9$$

From DP to CDCL

1960 Davis and Putnam algorithm : based on resolution rule , forget one variable after the other

Resolution rule

$$(x \vee y \vee \neg z) \otimes_x (\neg x \vee y \vee t) = y \vee \neg z \vee t$$

$$x_1 \vee x_4$$
$$x_1 \vee x_8 \vee x_{12}$$
$$x_1 \vee x_5 \vee \neg x_9$$

$$\neg x_1 \vee x_4 \vee x_{14}$$
$$\neg x_1 \vee \neg x_3 \vee \neg x_8$$

$$x_2 \vee x_{11}$$
$$\neg x_3 \vee \neg x_7 \vee x_{13}$$
$$\neg x_3 \vee \neg x_7 \vee \neg x_{13} \vee x_9$$
$$x_8 \vee \neg x_7 \vee \neg x_9$$

From DP to CDCL

1960 Davis and Putnam algorithm : based on resolution rule , forget one variable after the other

Resolution rule

$$(x \vee y \vee \neg z) \otimes_x (\neg x \vee y \vee t) = y \vee \neg z \vee t$$

$$x_4 \vee x_{14}$$
$$x_4 \vee \neg x_3 \vee \neg x_8$$
$$x_8 \vee x_{12} \vee x_4 \vee x_{14}$$
$$x_5 \vee \neg x_9 \vee x_4 \vee x_{14}$$
$$x_5 \vee \neg x_9 \vee \neg x_3 \vee \neg x_8$$

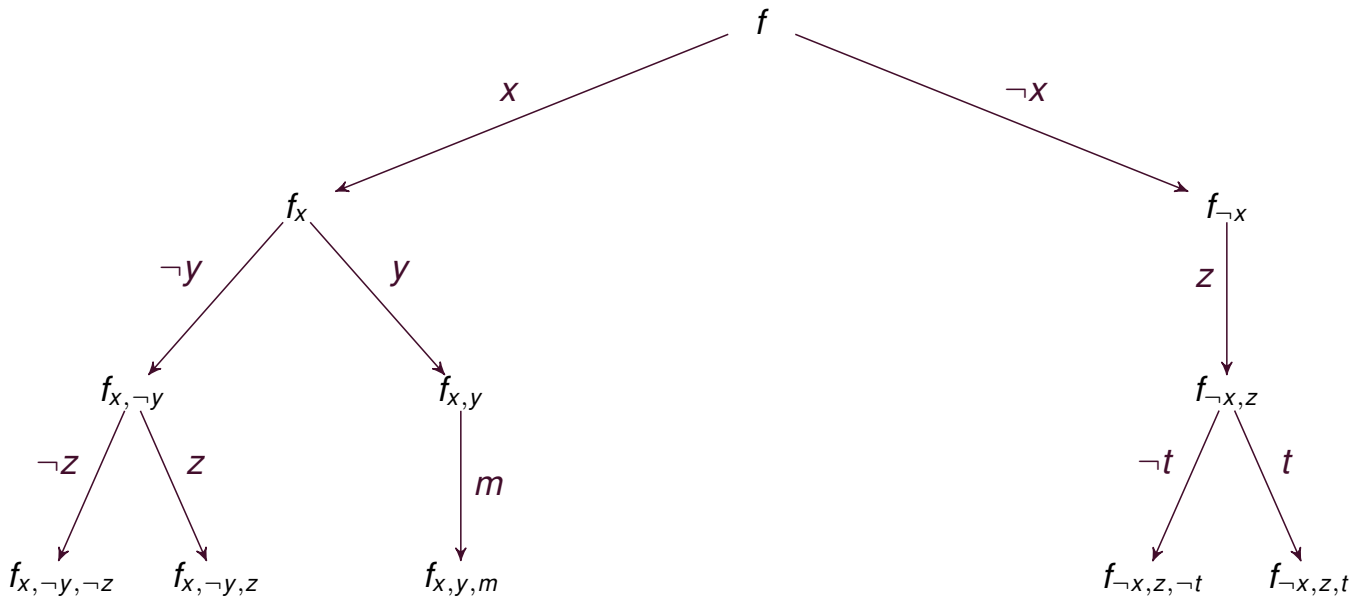
$$x_2 \vee x_{11}$$
$$\neg x_3 \vee \neg x_7 \vee x_{13}$$
$$\neg x_3 \vee \neg x_7 \vee \neg x_{13} \vee x_9$$
$$x_8 \vee \neg x_7 \vee \neg x_9$$

- Combinatorial explosion
- Used in pre-processing (See Marijn talk's)

From DP to CDCL

1960 Davis and Putnam algorithm : based on resolution, forget one variable after the other

1962 Davis, Logeman Loveland algorithm : backtrack search tree (partial models)



From DP to CDCL

1960	Davis and Putnam algorithm : based on resolution, forget one variable after the other
1962	Davis, Logeman Loveland algorithm : backtrack search tree (partial models)
1962 – 2001	Improve DPLL solvers : heuristics, look-ahead strategies...

From DP to CDCL

1960	Davis and Putnam algorithm (DP) : based on resolution, forget one variable after the other
1962	Davis, Logeman Loveland algorithm(DLL, DPLL) : backtrack search tree (partial models)
1962 – 2001	Improve DPLL solvers : heuristics, look-ahead strategies...
1992	"*Planning as Satisfiability*". Kautz and Selman. ECAI 92
1999	"*Symbolic Model Checking Without BDD*". Biere, Cimatti, Clarke and Zhu. TACAS 1999

- Encoding of application problems in SAT

From DP to CDCL

1960	Davis and Putnam algorithm (DP) : based on resolution, forget one variable after the other
1962	Davis, Logeman Loveland algorithm(DLL, DPLL) : backtrack search tree (partial models)
1962 – 2001	Improve DPLL solvers : heuristics, look-ahead strategies...
1992	"*Planning as Satisfiability*". Kautz and Selman. ECAI
1999	"*Symbolic Model Checking Without BDD*". Biere, Cimatti, Clarke and Zhu. TACAS
1996	"*GRASP - a new search algorithm for satisfiability*".Sakallah and Marques-Silva. ICCAD
2001	"*Chaff : Engineering an Efficient SAT Solver*". Moskewicz, Madigan, Zhao, Zhang and Malik. DAC
2003	"*An extensible SAT solver*". Een and Sorensson. SAT

- Learning + dedicated data structures for managing huge formulas

Before 2000

- Look Ahead solvers

  To know where to go, you need to know where you are

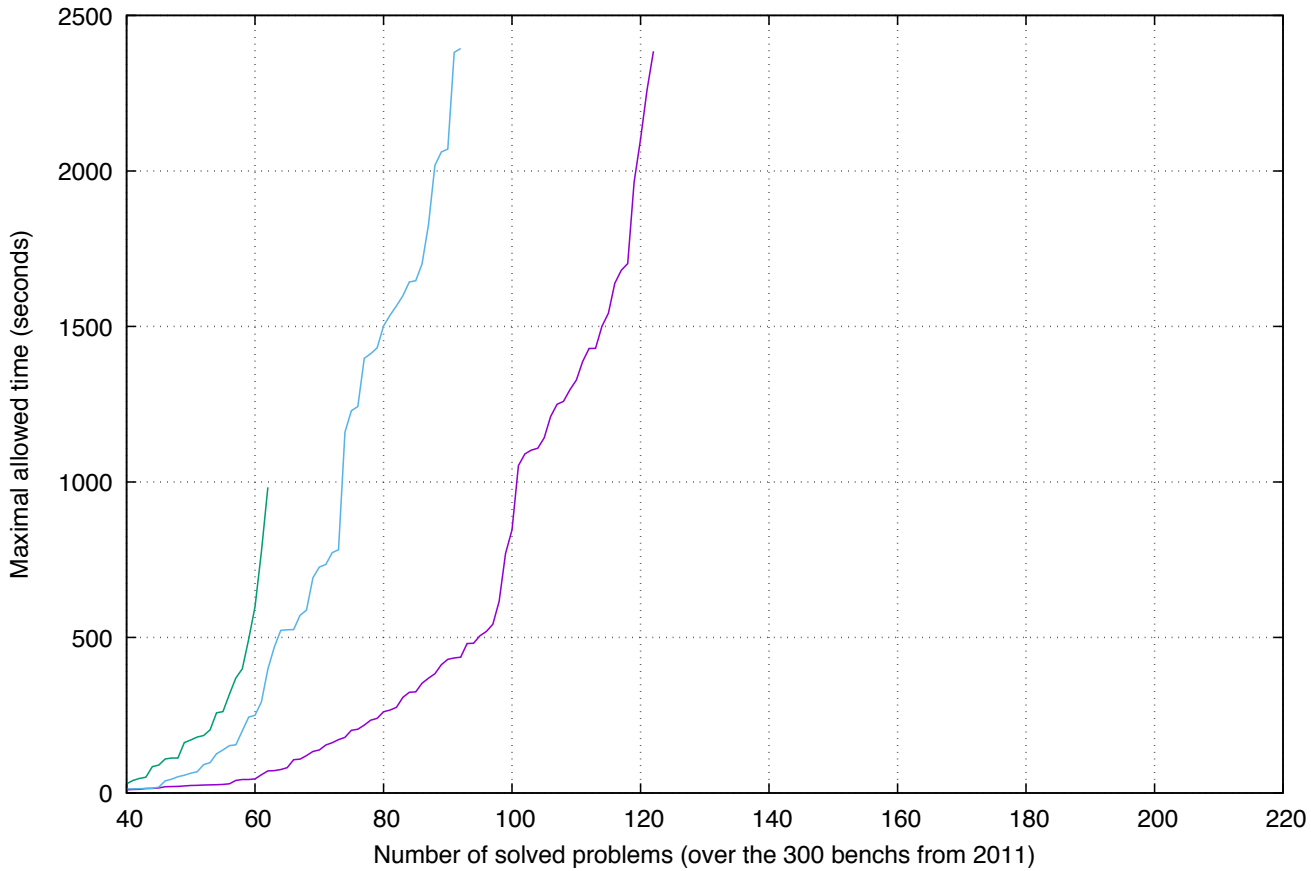
  - Search tree
  - Ideas/Techniques that aim to reduce/balance the search tree.
  - Heuristics, failed literals
  - Explanation of performances is (relatively) simple

- Lookback solvers (CDCL)

  You do not know where you are, but you know where to go

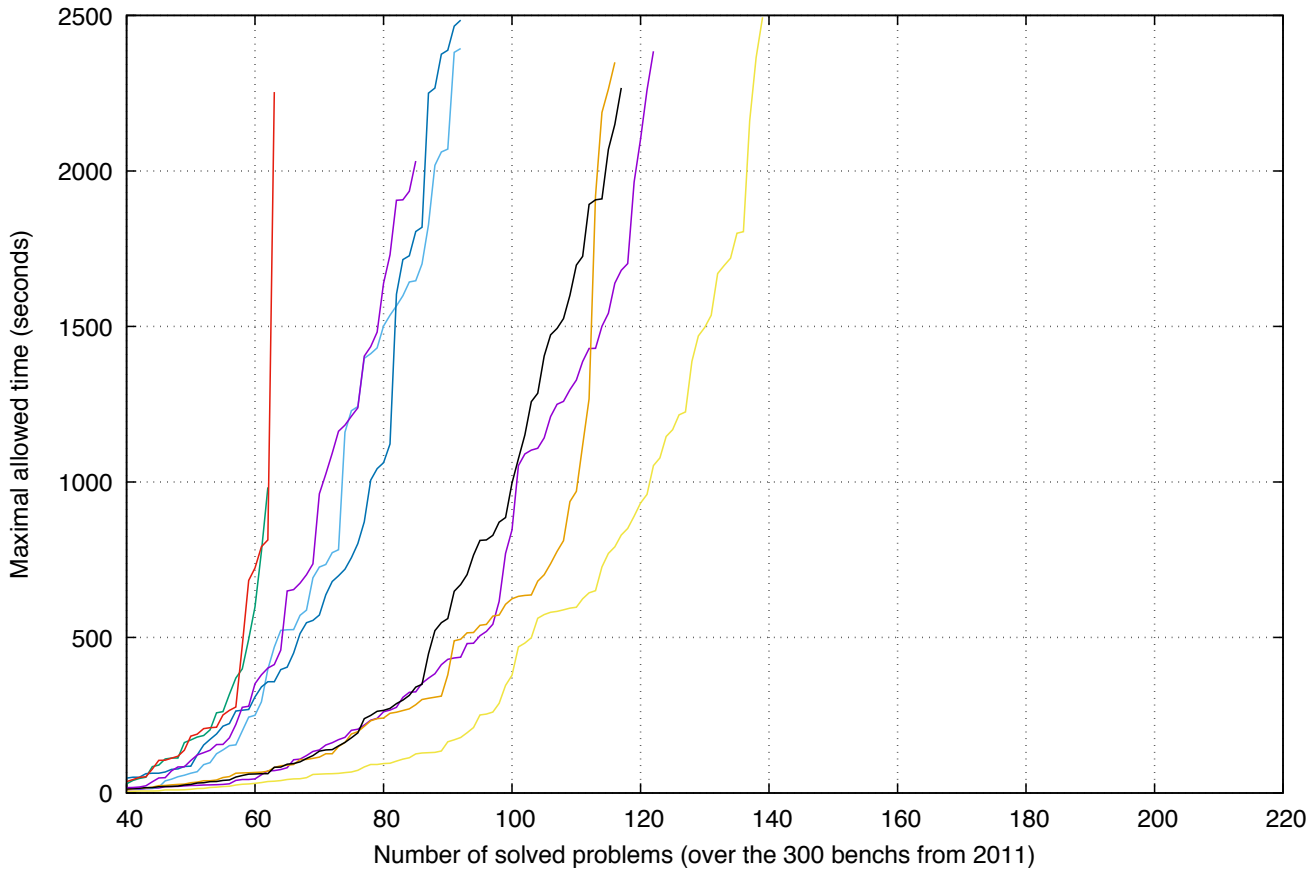
  - Many variables, many clauses
  - Restarts
  - Explanation of performances is quite hard

Performances after 2001



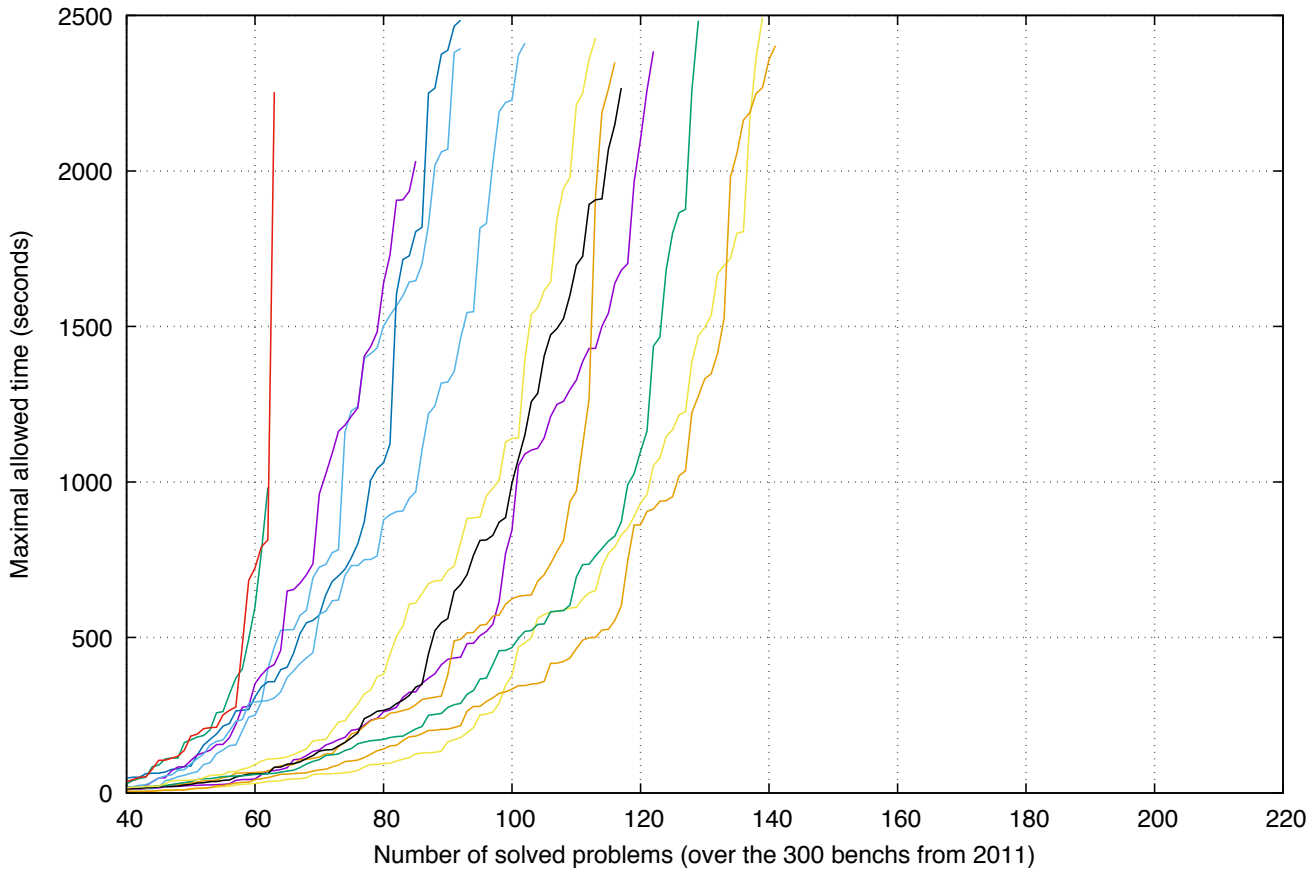
2002

Performances after 2001



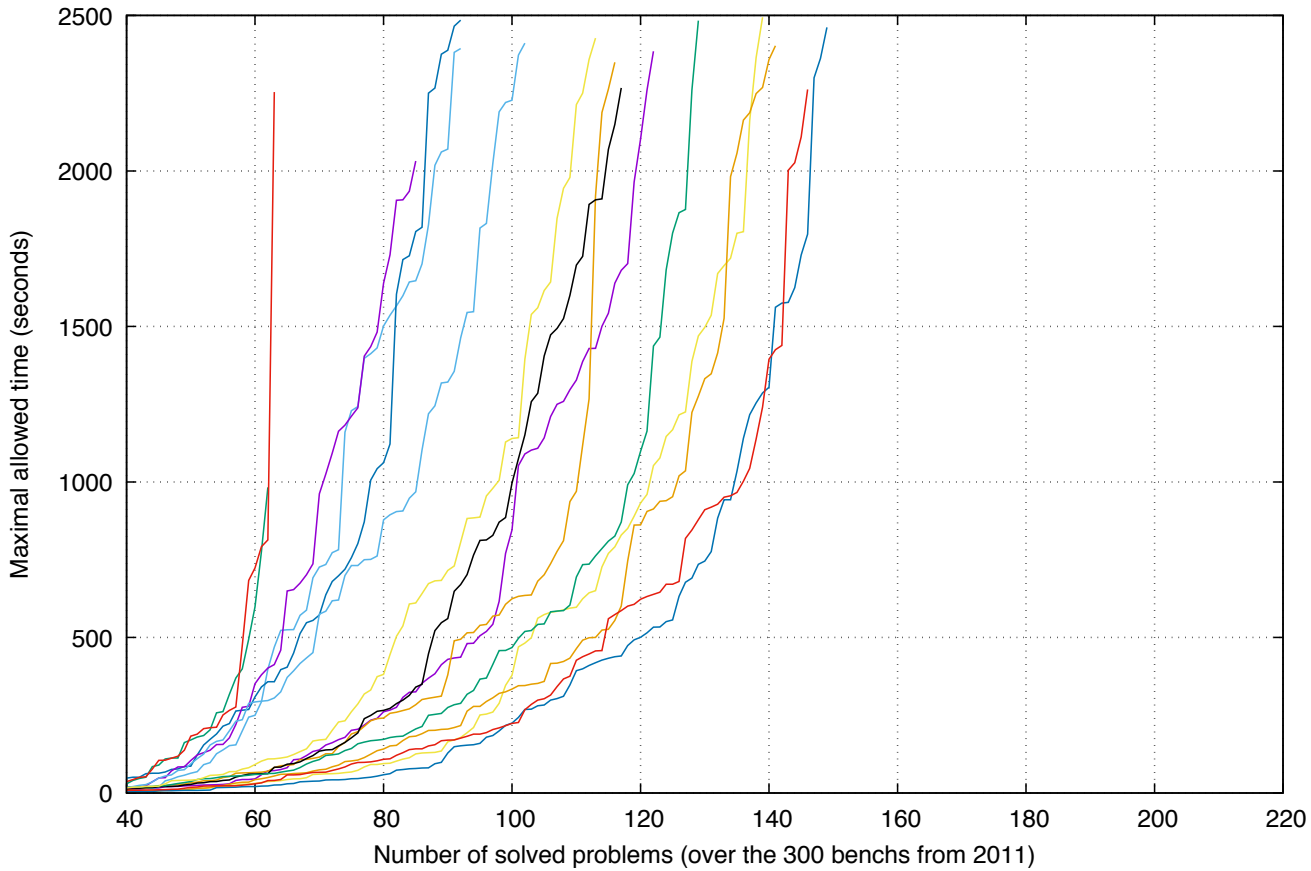
2003

Performances after 2001



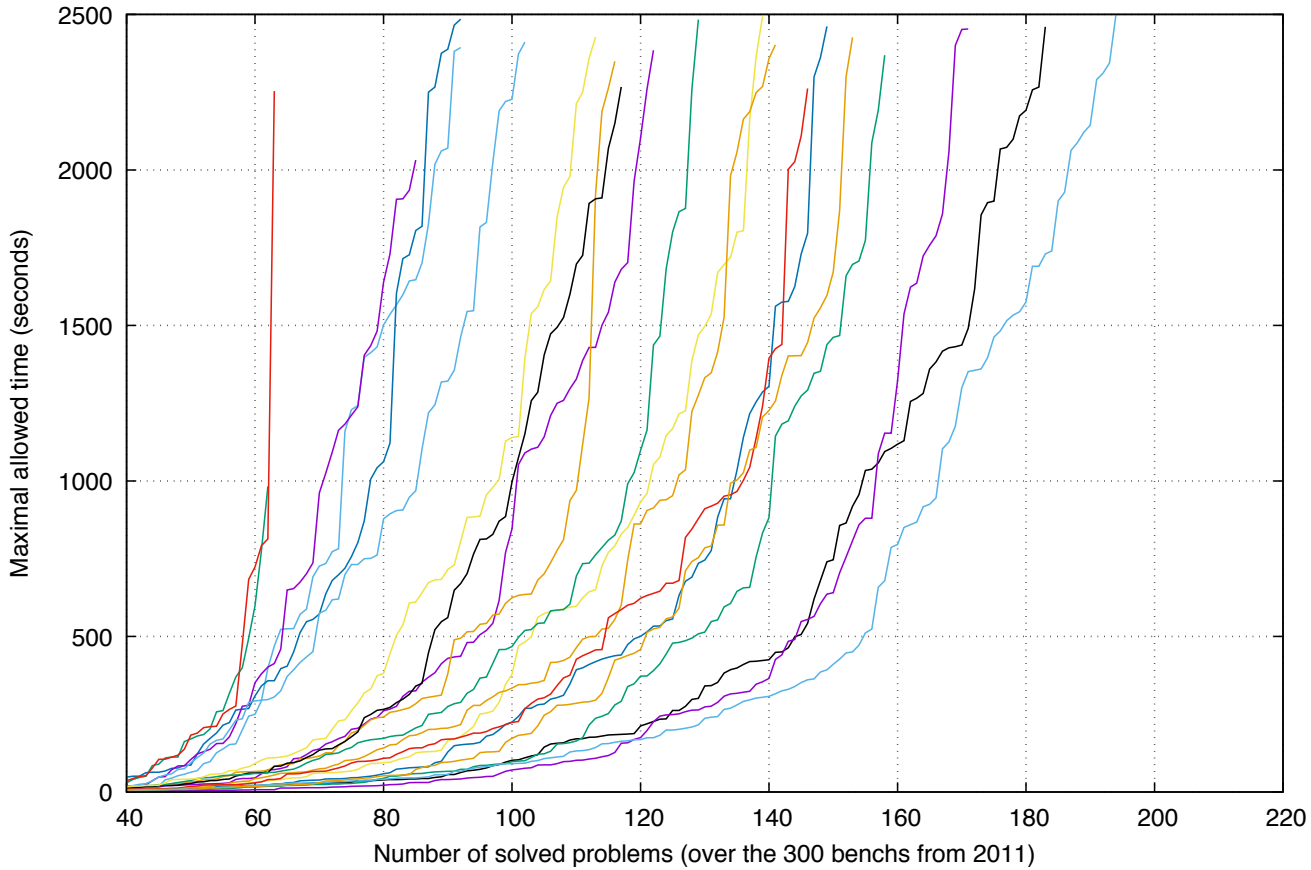
2005

Performances after 2001



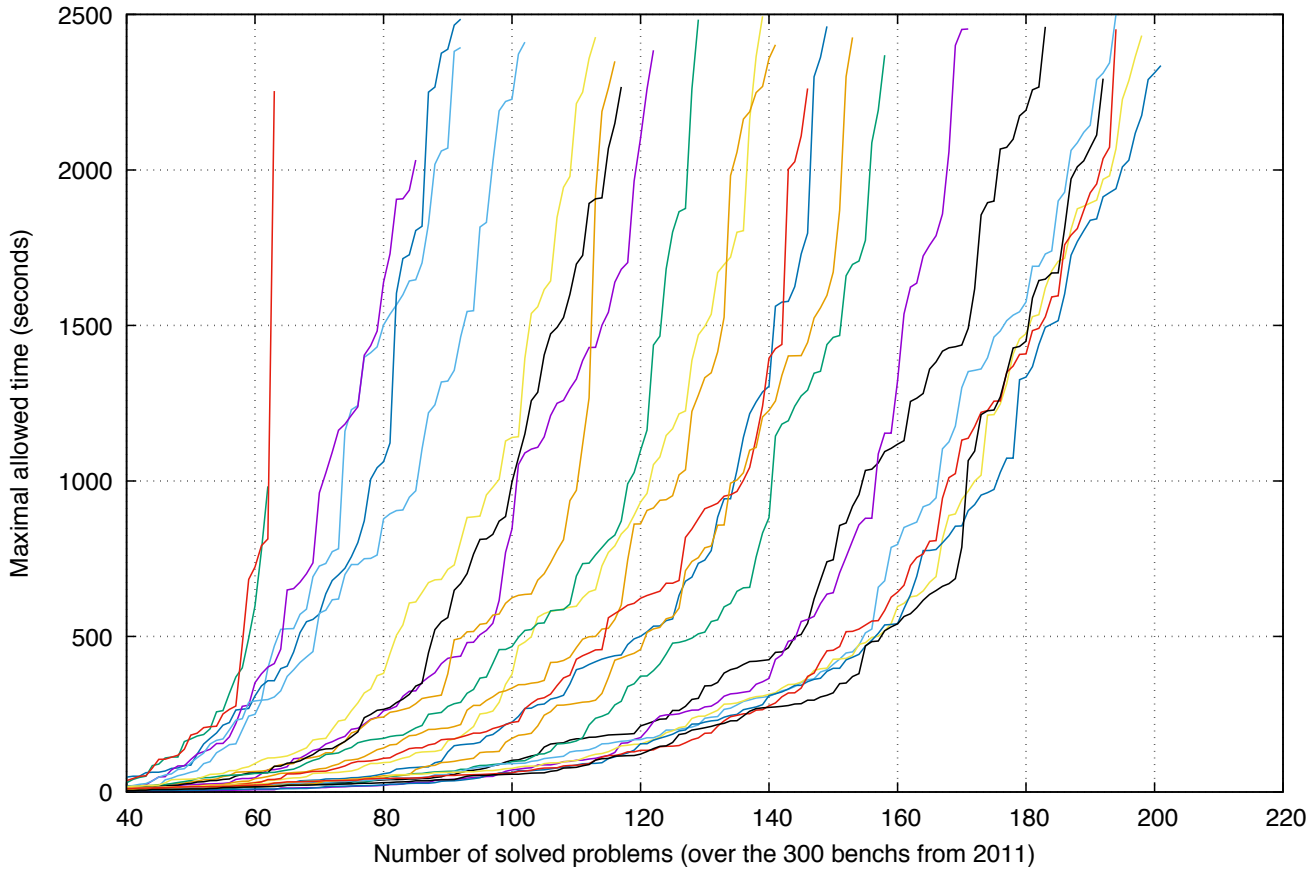
2007

Performances after 2001



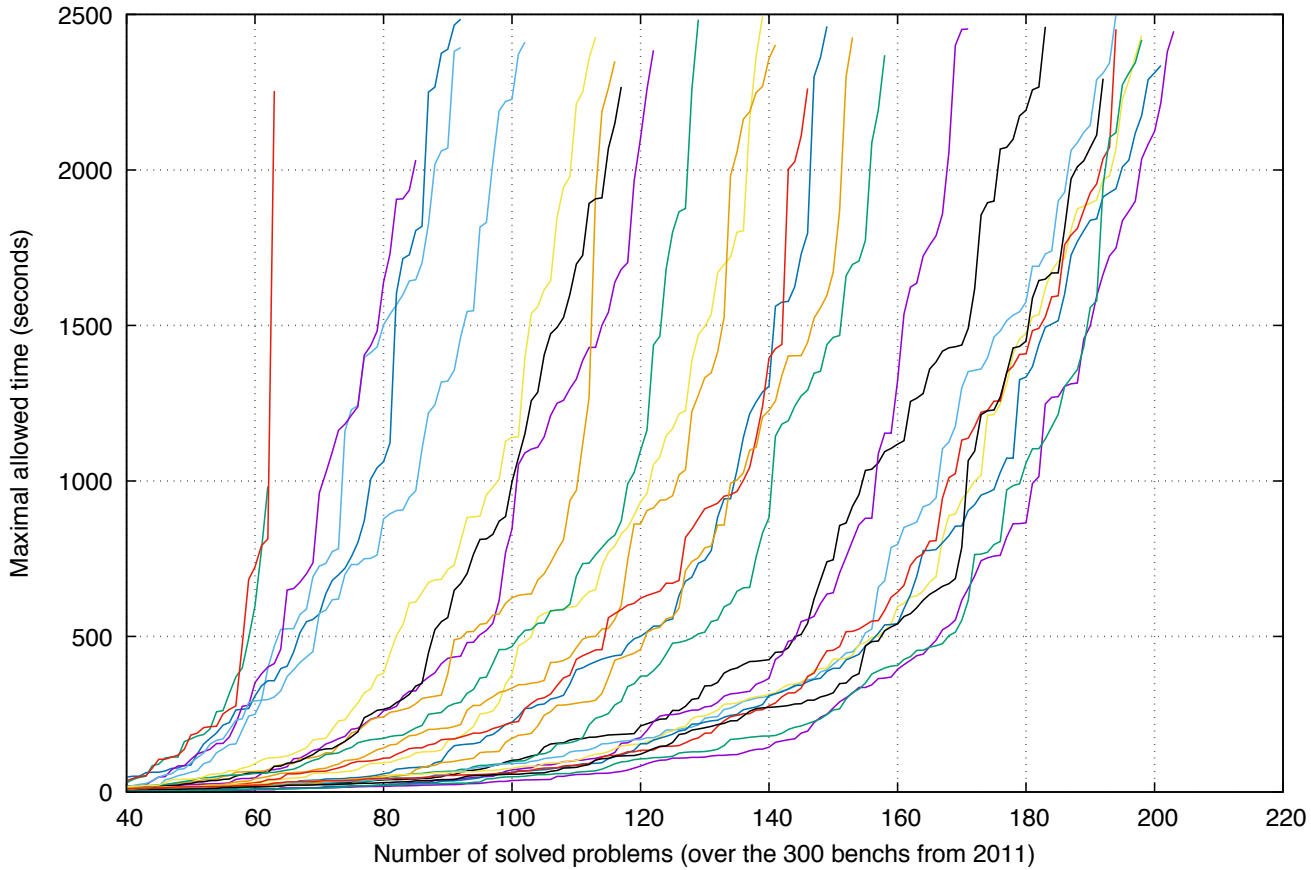
2009

Performances after 2001



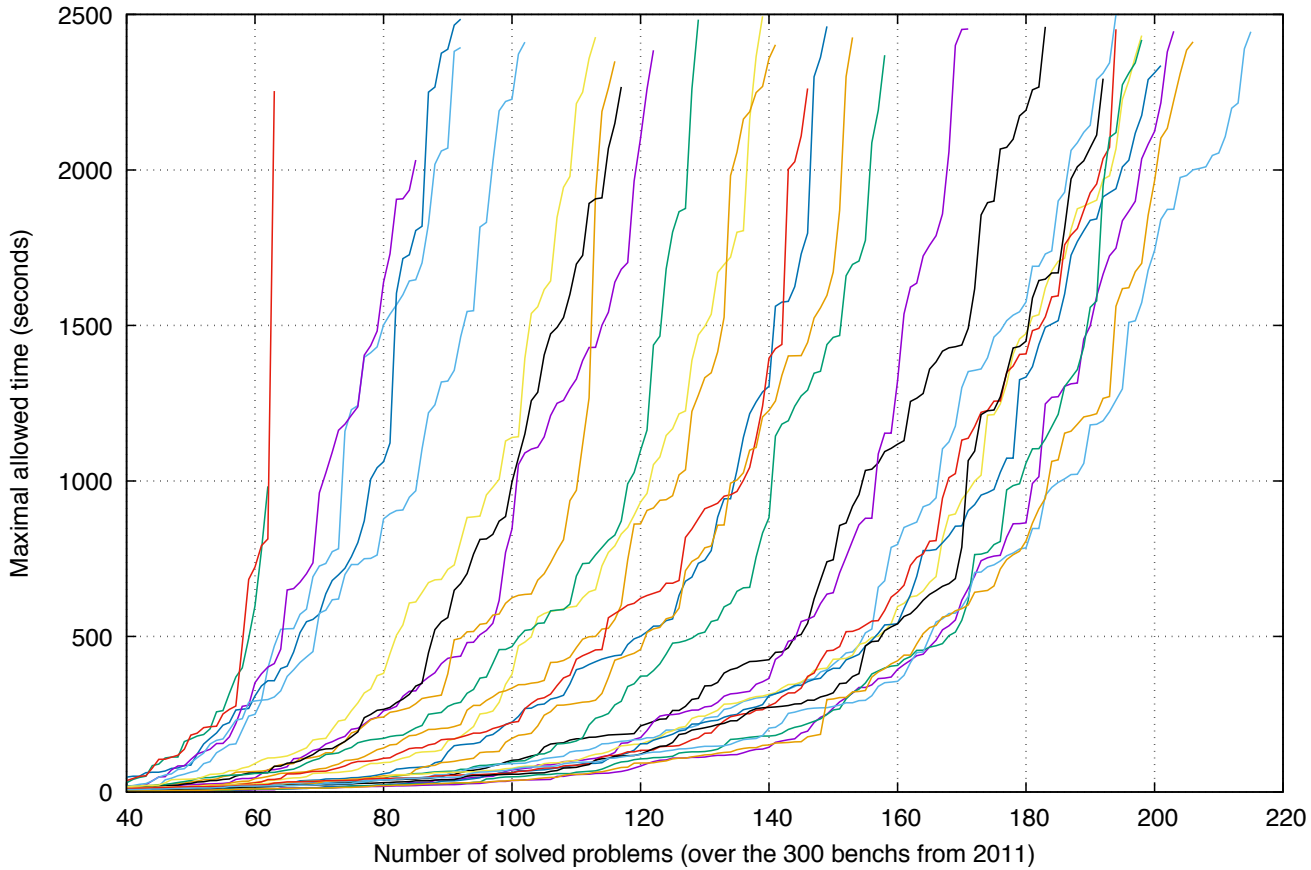
2011

Performances after 2001

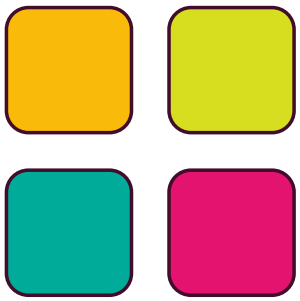


2014

Performances after 2001



2016



CDCL Solvers

CDCL Architecture

A Short Overview of CDCL Solvers

Sequence of decision, propagations

$$
\begin{aligned}
c_1 &= x_1 \vee x_4 \\
c_2 &= x_1 \vee \overline{x_3} \vee \overline{x_8} \\
c_3 &= x_1 \vee x_8 \vee x_{12} \\
c_4 &= x_2 \vee x_{11} \\
c_5 &= \overline{x_3} \vee \overline{x_7} \vee x_{13} \\
c_6 &= \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9 \\
c_7 &= x_8 \vee \overline{x_7} \vee \overline{x_9}
\end{aligned}
$$

A Short Overview of CDCL Solvers

Sequence of decision, propagations

DL 1

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

A Short Overview of CDCL Solvers

Sequence of decision, propagations

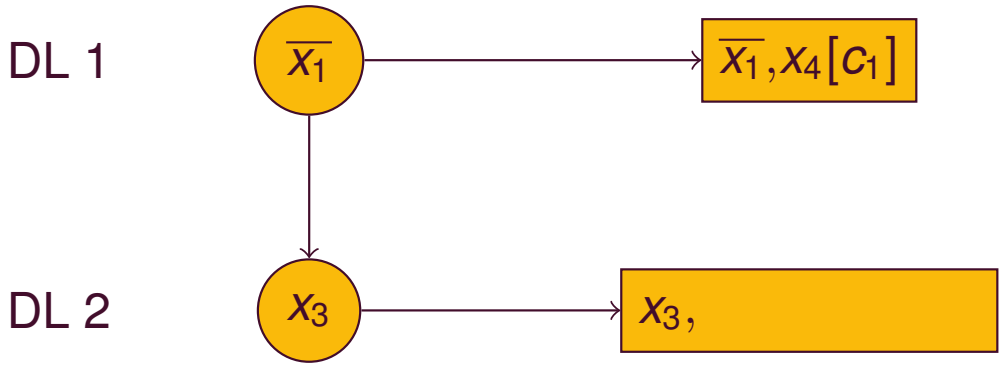
DL 1

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

A Short Overview of CDCL Solvers

Sequence of decision, propagations

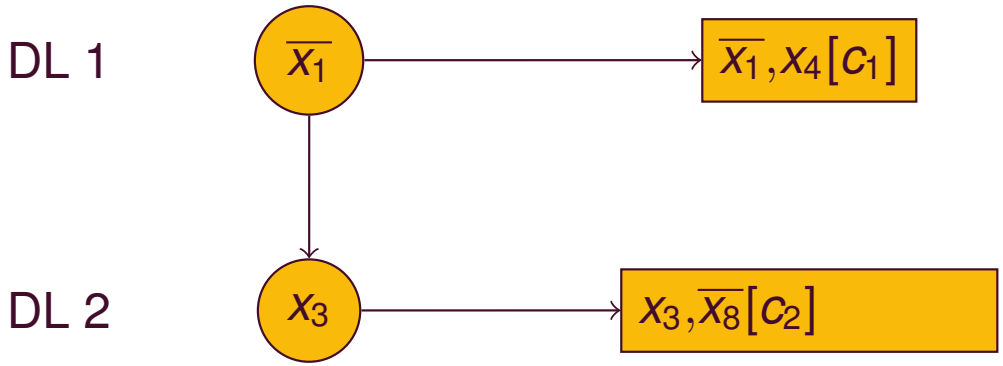
$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$



A Short Overview of CDCL Solvers

Sequence of decision, propagations

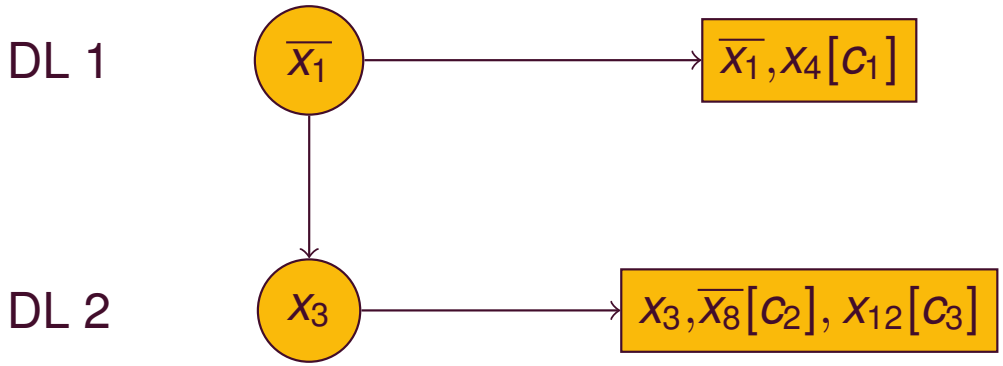
$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$



A Short Overview of CDCL Solvers

Sequence of decision, propagations

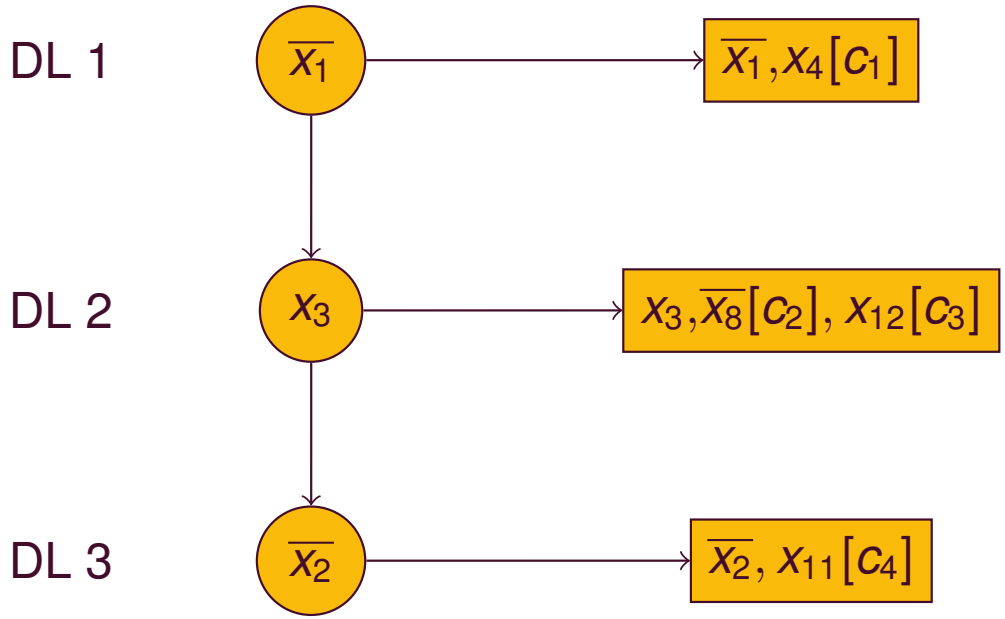
$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$



A Short Overview of CDCL Solvers

Sequence of decision, propagations

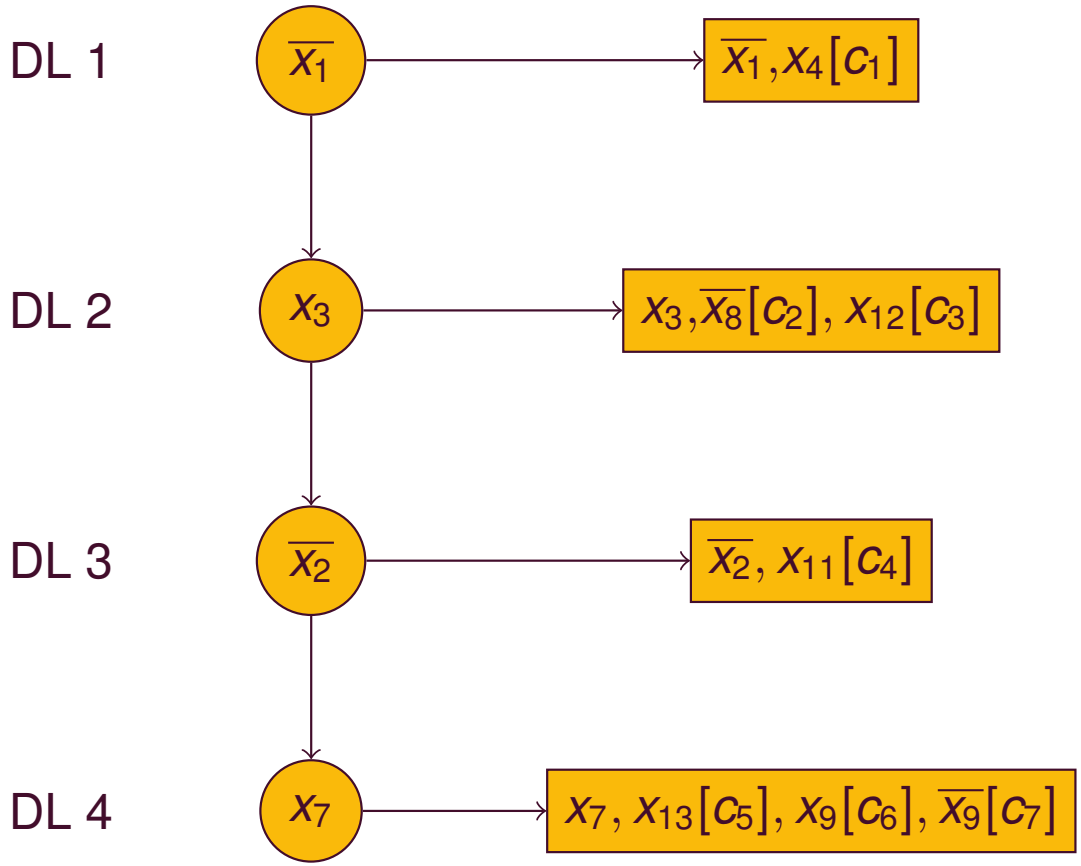
$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$



A Short Overview of CDCL Solvers

Sequence of decision, propagations

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

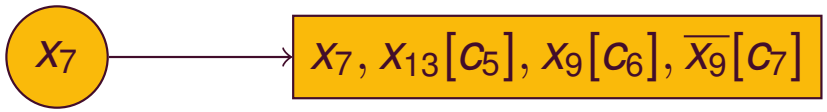


A Short Overview of CDCL Solvers

Conflict analysis

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

DL 4

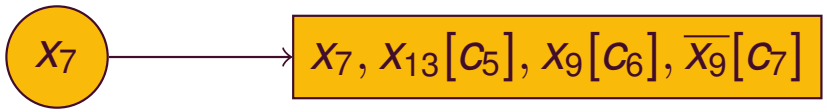


A Short Overview of CDCL Solvers

Conflict analysis

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

DL 4



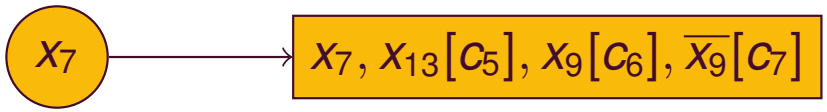
$$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$$

A Short Overview of CDCL Solvers

Conflict analysis

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

DL 4



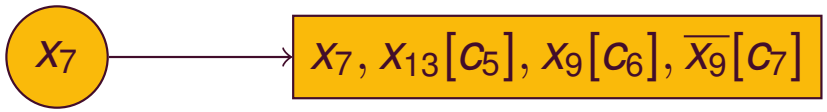
$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$
$d_1 = d^* \otimes_{x_{13}} c_5 = \overline{x_3} \vee x_8 \vee \overline{x_7}$

A Short Overview of CDCL Solvers

Conflict analysis

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

DL 4



$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$
$d_1 = d^* \otimes_{x_{13}} c_5 = \overline{x_3} \vee x_8 \vee \overline{x_7}$

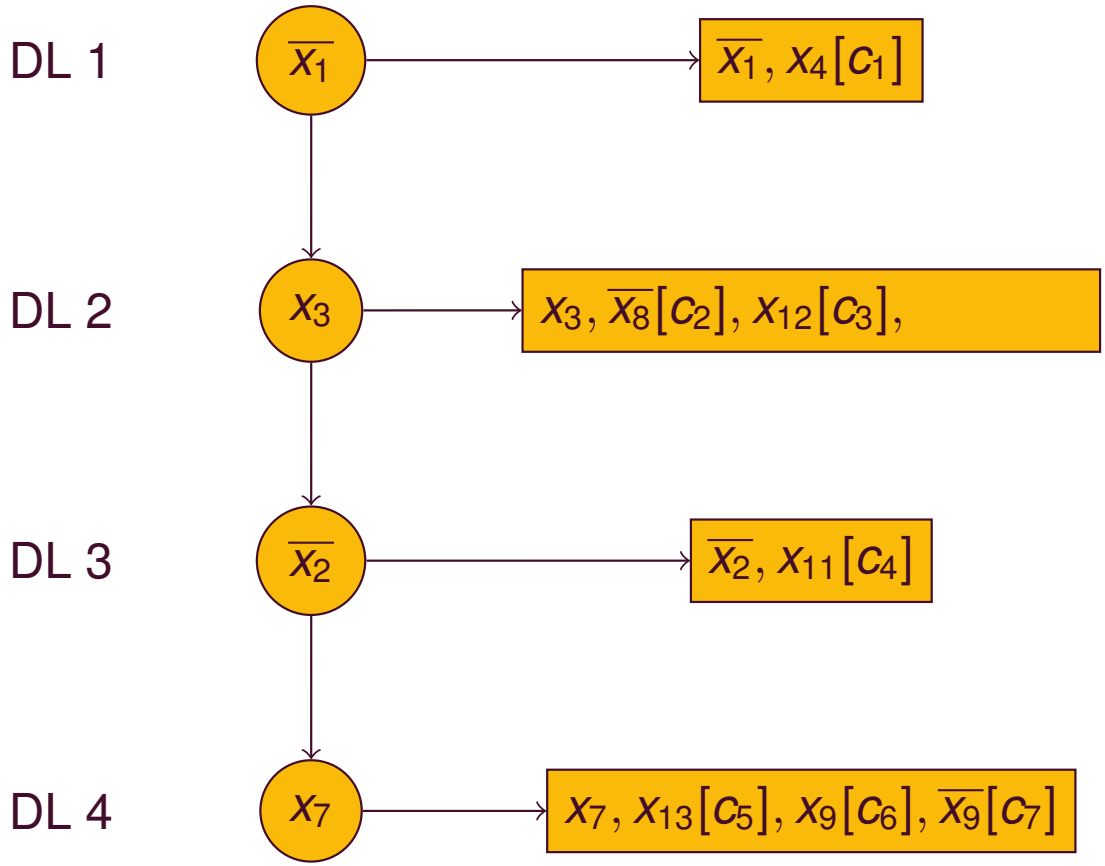
- First resolvent that contains only one literal of the last decision level
- First UIP learning scheme (related to implication graph)
- d_1 is added to the formula and...

A Short Overview of CDCL Solvers

Backjumping

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$

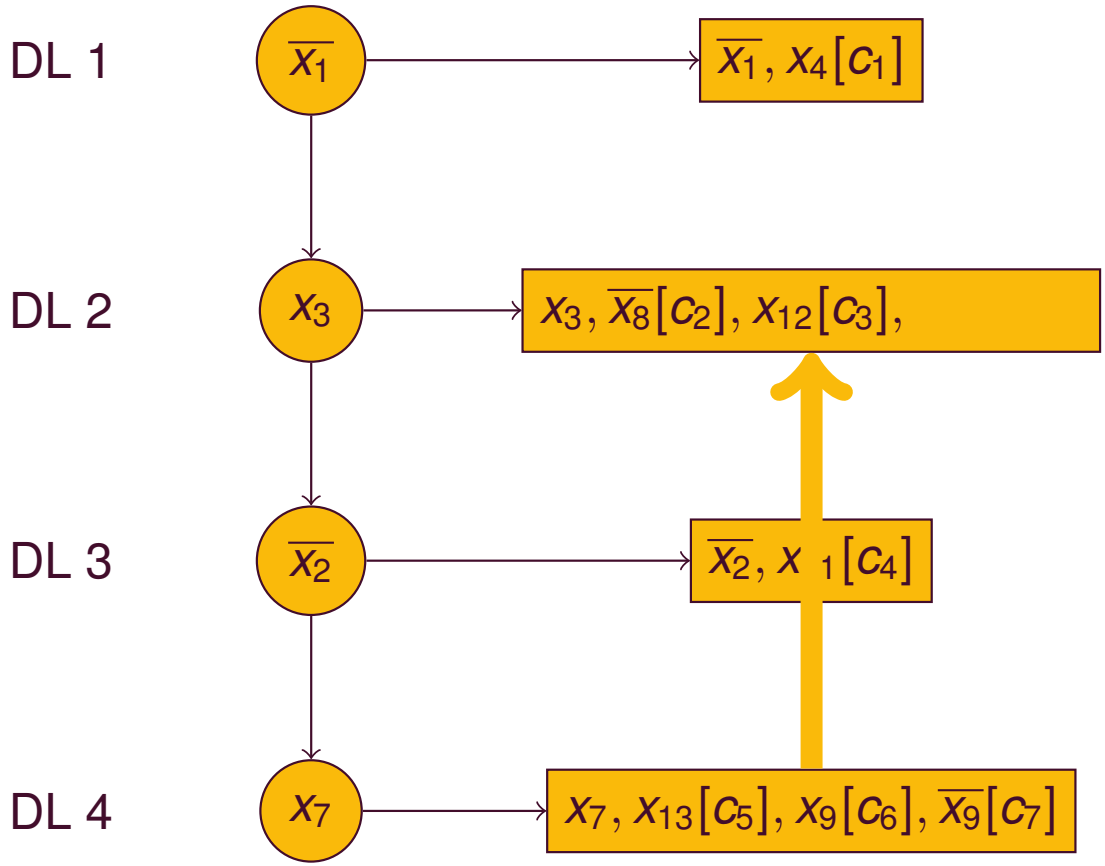


A Short Overview of CDCL Solvers

Backjumping

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$

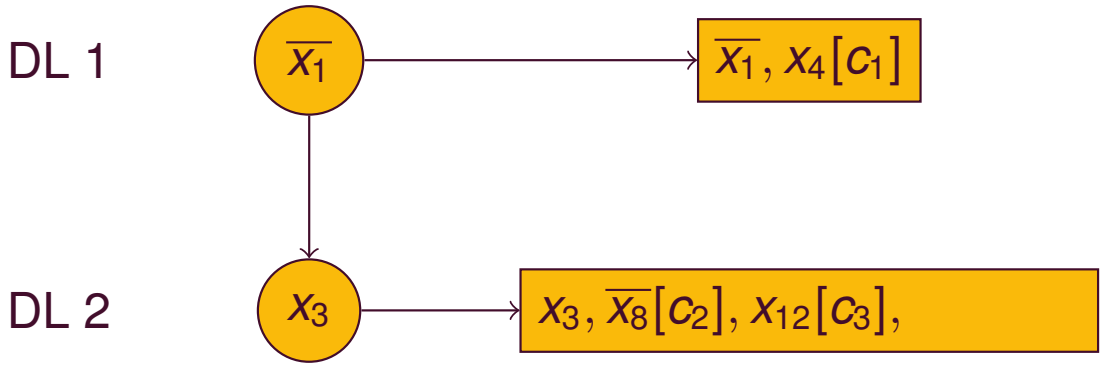


A Short Overview of CDCL Solvers

Backjumping

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$

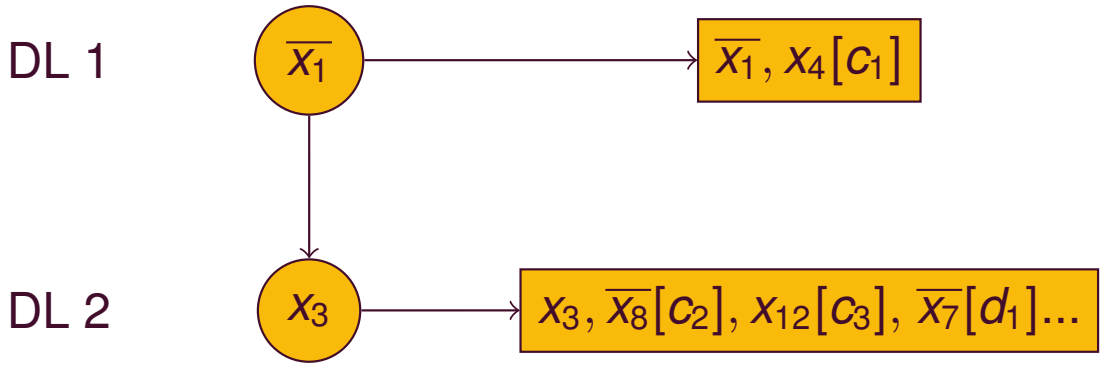


A Short Overview of CDCL Solvers

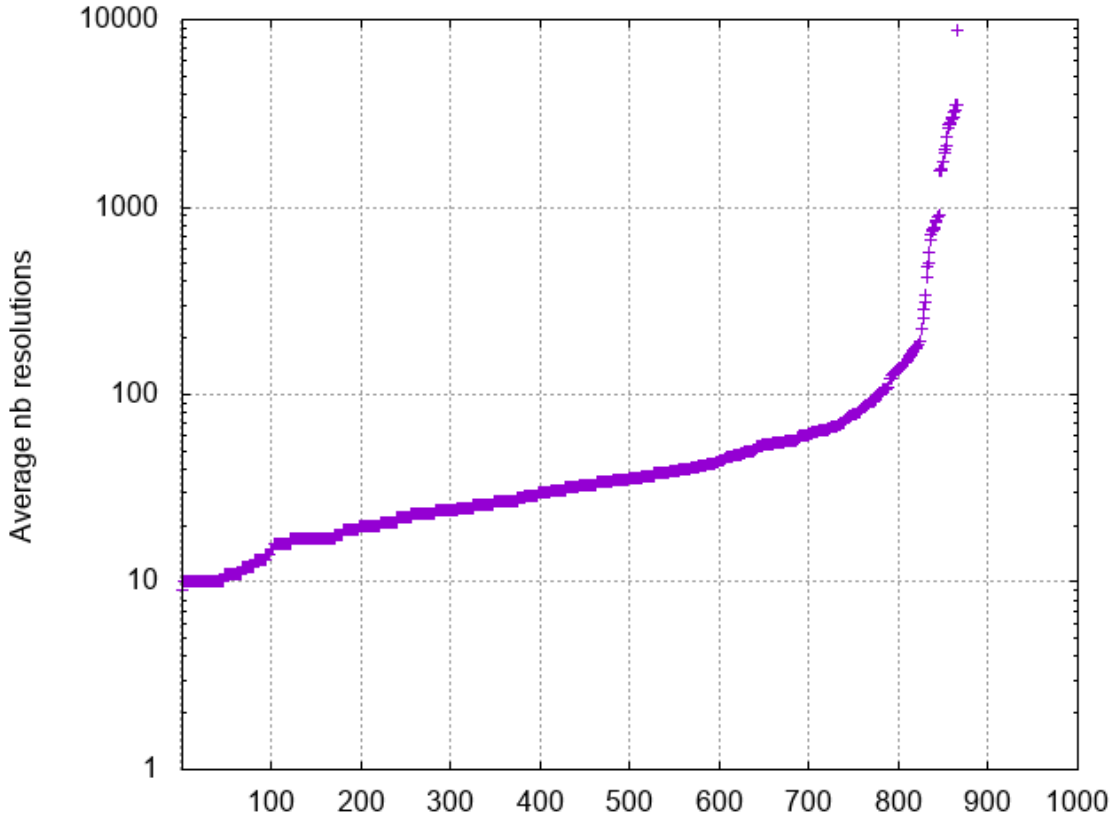
Backjumping

$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$

$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$



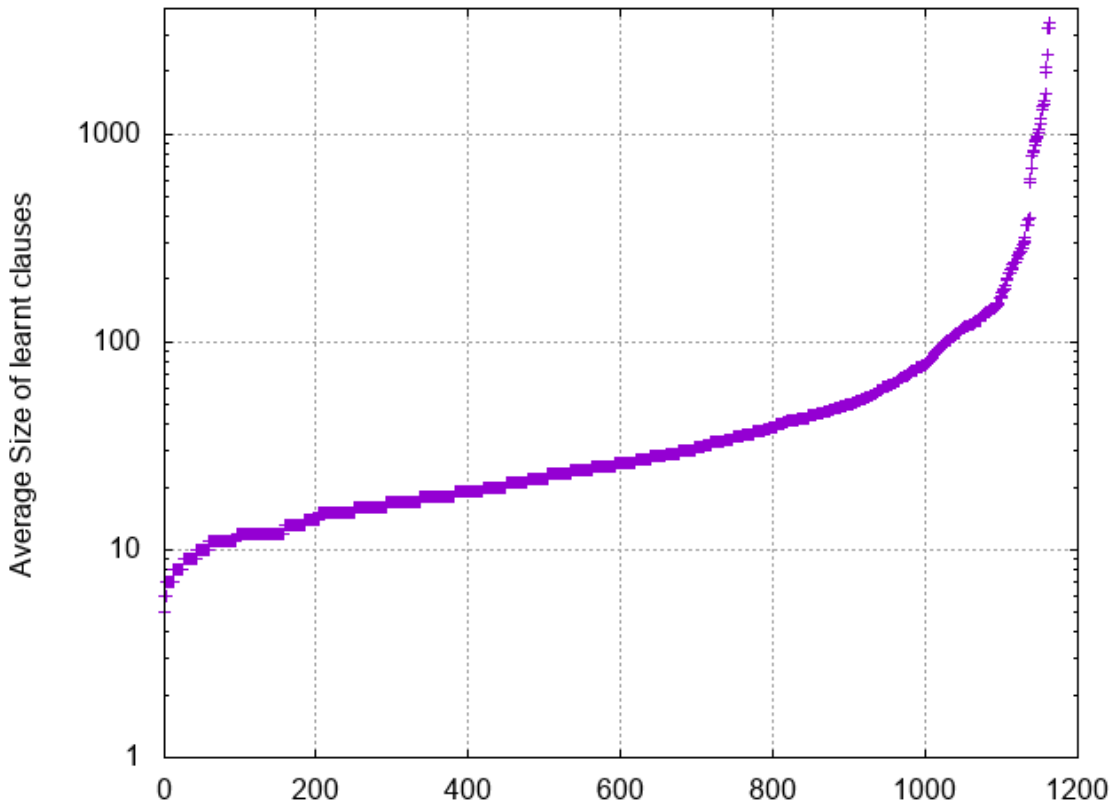
Many resolutions



stats after 10,000 conflicts on 900 benchmarks

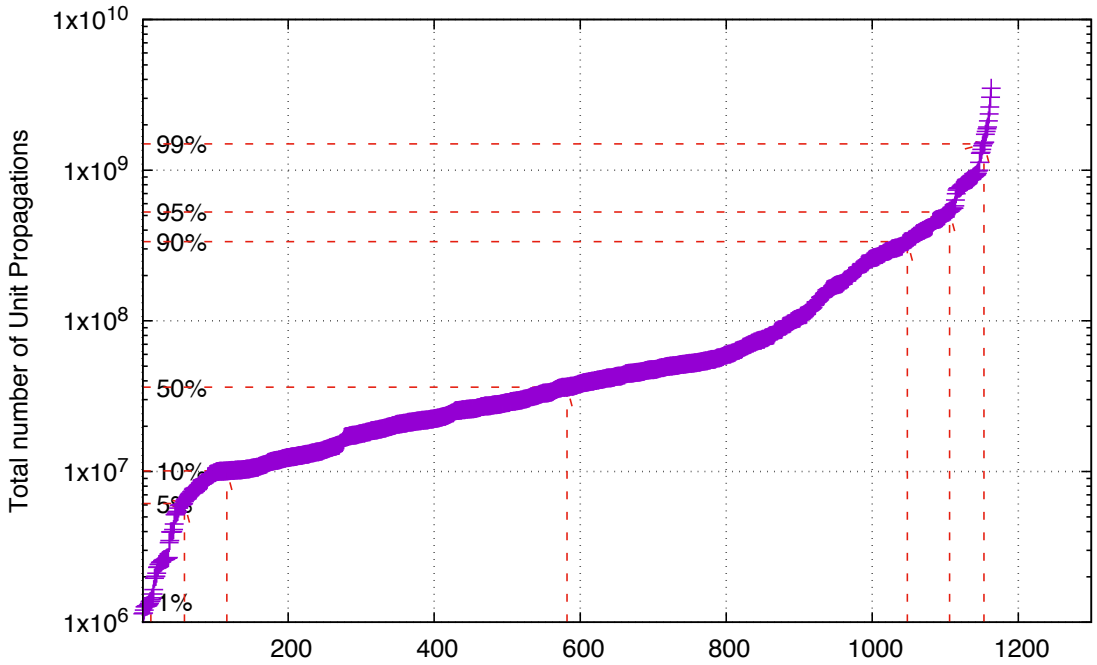
- Many resolutions on correlated clauses
- Only one clause stored !
- One of the reason of the efficiency of learning

Long clauses



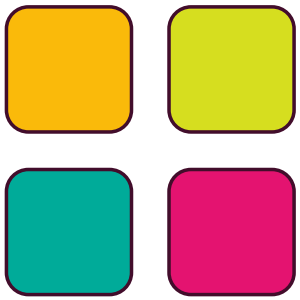
stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

A lot of propagations



stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

- We need to pay attention on BCP engine



Unit Propagation – Watched Literals

Storing occurences

- We need to know when a clause is unit or in conflict
- Each time a propagation is done the formula is reduced
- Learning : many (potentially large) clauses
- Before CDCL solvers, all occurences of literals were stored

$c_1 = x_1 \vee x_4$
$c_2 = \neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
$c_3 = x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5$
$c_4 = x_2 \vee x_5 \vee \neg x_6$
...

$x_1 \rightarrow \{c_1, c_3\}$
$\neg x_1 \rightarrow \{c_2\}$

$x_2 \rightarrow \{c_2, c_4\}$
$\neg x_2 \rightarrow \{c_3\}$

$x_3 \rightarrow \{c_2\}$
$\neg x_3 \rightarrow \{c_3\}$

$x_4 \rightarrow \{c_1, c_2, c_3\}$
$\neg x_4 \rightarrow \{\}$

$x_5 \rightarrow \{c_2, c_4\}$
$\neg x_5 \rightarrow \{c_3\}$

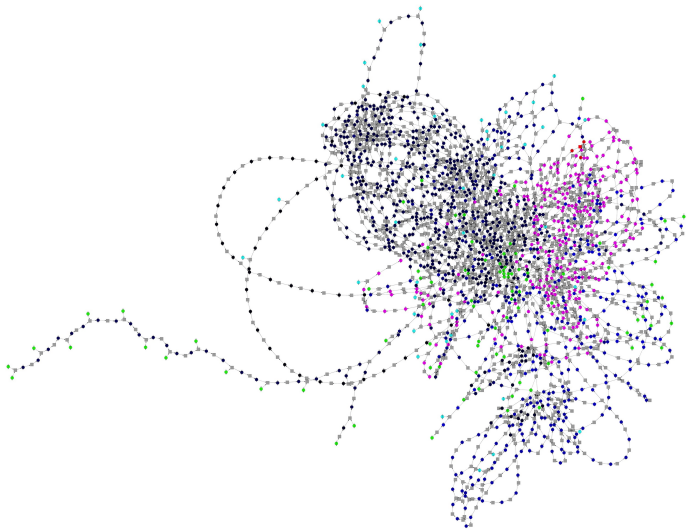
$x_6 \rightarrow \{\}$
$\neg x_6 \rightarrow \{c_4\}$

- Store the current size of the clause and the current state
- When propagating x_4, take a look to all clauses where $\neg x_4$ appears (c_1, c_2, c_3
- Update size in consequence
- When backtracking, restore counters

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

- The first two literals are the witnesses

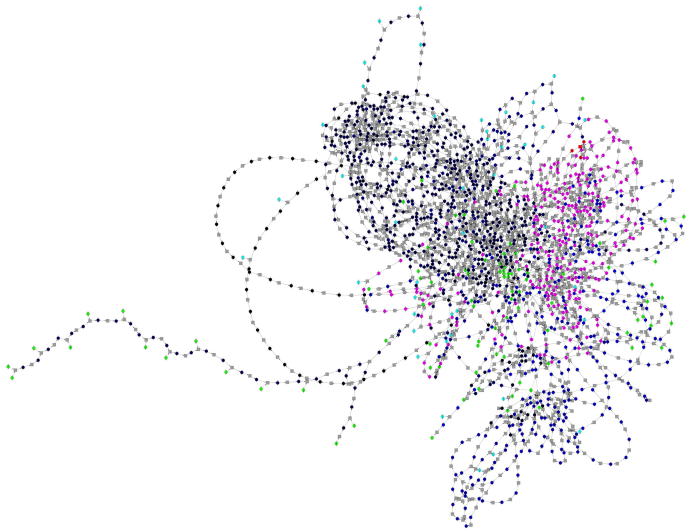


Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do

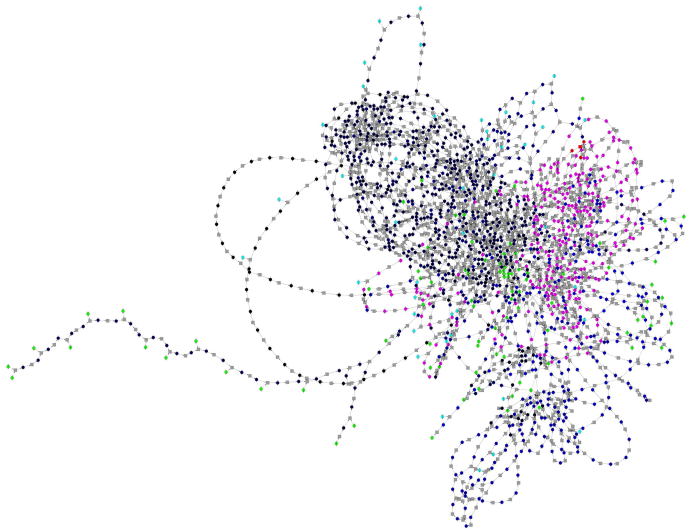


Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause

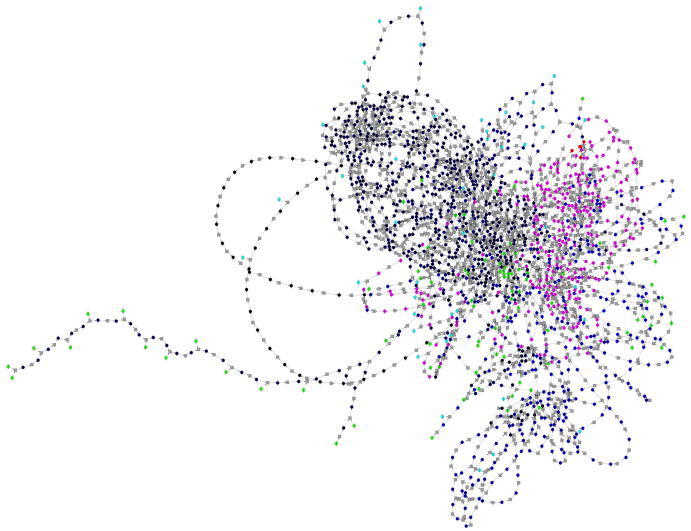


Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause



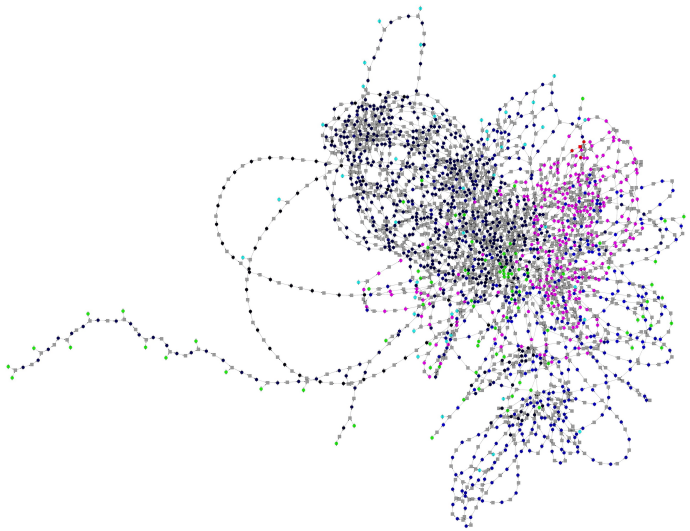
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause



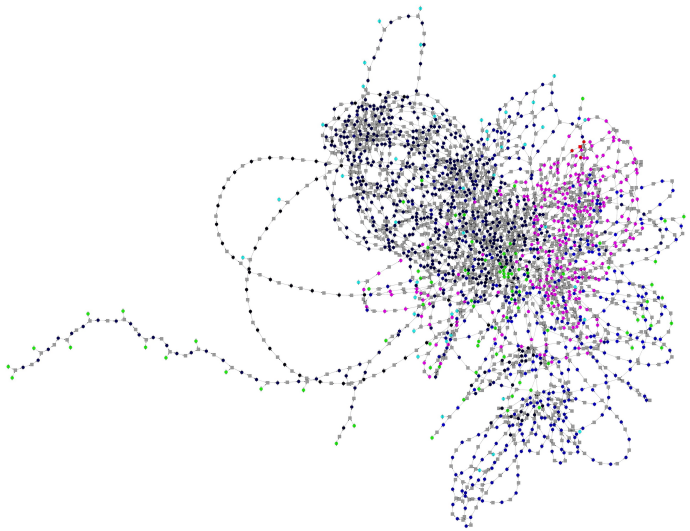
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause
- ℓ_5 is falsified : nothing to do



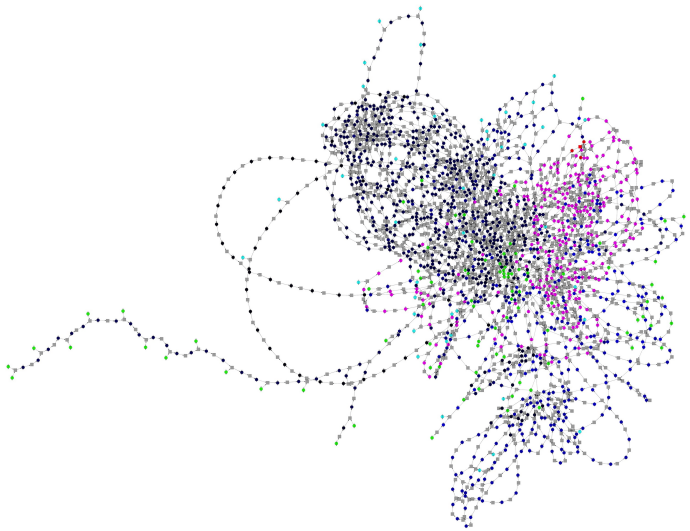
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause
- ℓ_5 is falsified : nothing to do
- ℓ_1 is falsified : need to find a new watch. None exists. ℓ_4 is unit. We have to propagate it.



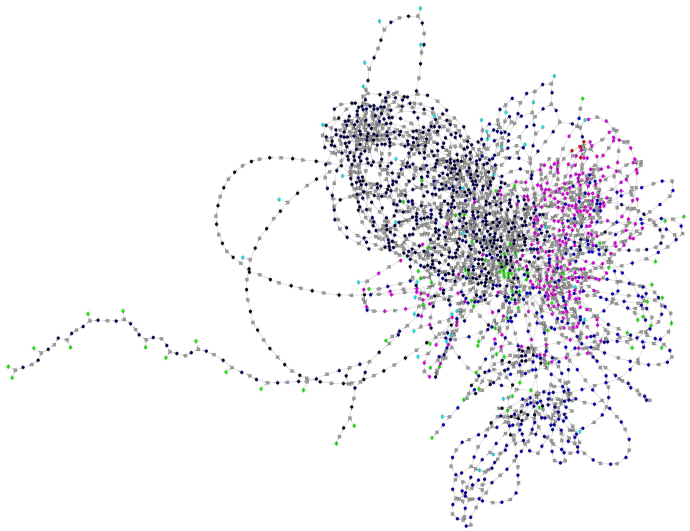
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause
- ℓ_5 is falsified : nothing to do
- ℓ_1 is falsified : need to find a new watch. None exists. ℓ_4 is unit. We have to propagate it.
- Backtrack : nothing to do ! ! ! ! !



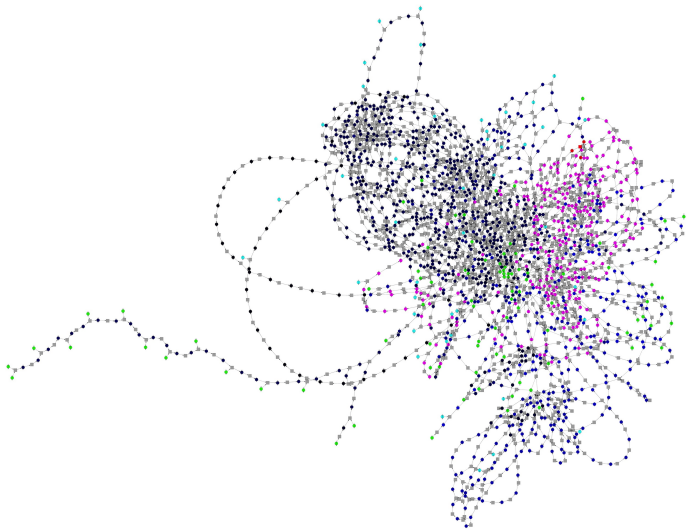
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause
- ℓ_5 is falsified : nothing to do
- ℓ_1 is falsified : need to find a new watch. None exists. ℓ_4 is unit. We have to propagate it.



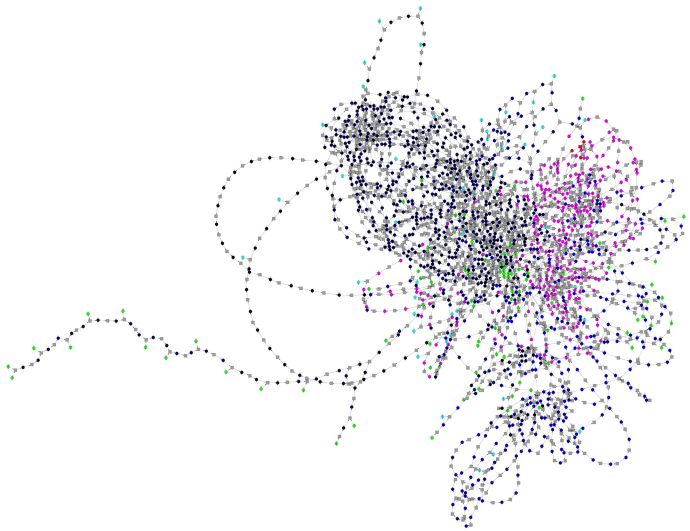
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause
- ℓ_5 is falsified : nothing to do



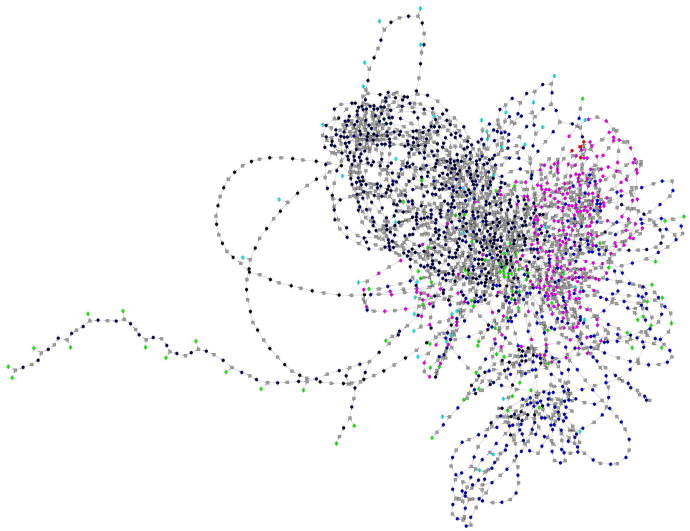
Real Implication Graph

Be Lazy

- CHAFF introduced a lazy data-structure to perform unit propagation efficiently
- Two watchers insuring that a clause is at least binary
- Nothing to do if the clause is satisfied
- Traverse the clause only if one of the watcher becomes false
- Backtrack is free (no need to traverse old unit clauses)

ℓ_1	ℓ_4	ℓ_3	ℓ_2	ℓ_5

- The first two literals are the witnesses
- ℓ_3 is falsified : nothing to do
- ℓ_2 is falsified : need to find a new watch, traverse the clause



Real Implication Graph

Running example

$$c_1 = x_1 \vee x_4$$
$$c_2 = \neg x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$$
$$c_3 = x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5$$
$$c_4 = x_2 \vee x_5 \vee \neg x_6$$
$$\ldots$$

$$x_1 \rightarrow \{c_1, c_3\} \qquad \neg x_1 \rightarrow \{c_2\}$$
$$x_2 \rightarrow \{c_2, c_4\} \qquad \neg x_2 \rightarrow \{c_3\}$$
$$x_3 \rightarrow \{\} \qquad \neg x_3 \rightarrow \{\}$$
$$x_4 \rightarrow \{c_1\} \qquad \neg x_4 \rightarrow \{\}$$
$$x_5 \rightarrow \{c_4\} \qquad \neg x_5 \rightarrow \{\}$$
$$x_6 \rightarrow \{\} \qquad \neg x_6 \rightarrow \{\}$$

- When x_1 is propagated, take a look to all clauses watched by $\neg x_1$ (here jsute C_2
- Check if the other watched is satisfied
- If not, check for a new watch

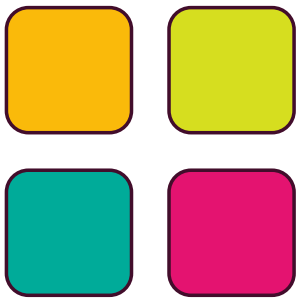
Need of Techniques that Support this Data-Structure

No knowledge of the current state of the formula

- ~~How many clauses are satisfied~~
- ~~How many binary clauses~~
- ~~Pure literals~~

- Only one guarantee : Unit clauses and conflicts are detected
- Model is discovered if all variables are assigned
- Heuristic for decision variable : we need to observe the past

- Take a look at a CDCL solver (the essentials of Minisat in fact) : addClause, Main loop, propagate, analyze



Heuristic

Decision Heuristic

- Goal : choose the next literal to assign
- Very important component
- Even more, without restarts : in such a case, first choices are essential !
- Before CDCL, it was based on the occurences in the formula (reduced) :
  - The more a variable appears in the formula, the better it is
  - Even more on (current) short clauses
- Not possible in CDCL
  - We do not know where variables occur
  - we do not know the current state of the formula
- Take into account the past

Favor variables that appears **recently** in conflict/analysis

- VSIDS : Introduced in Chaff
- EVSIDS : Exponential VSIDS, introduced in Minisat
- Must pay attention to data-structures

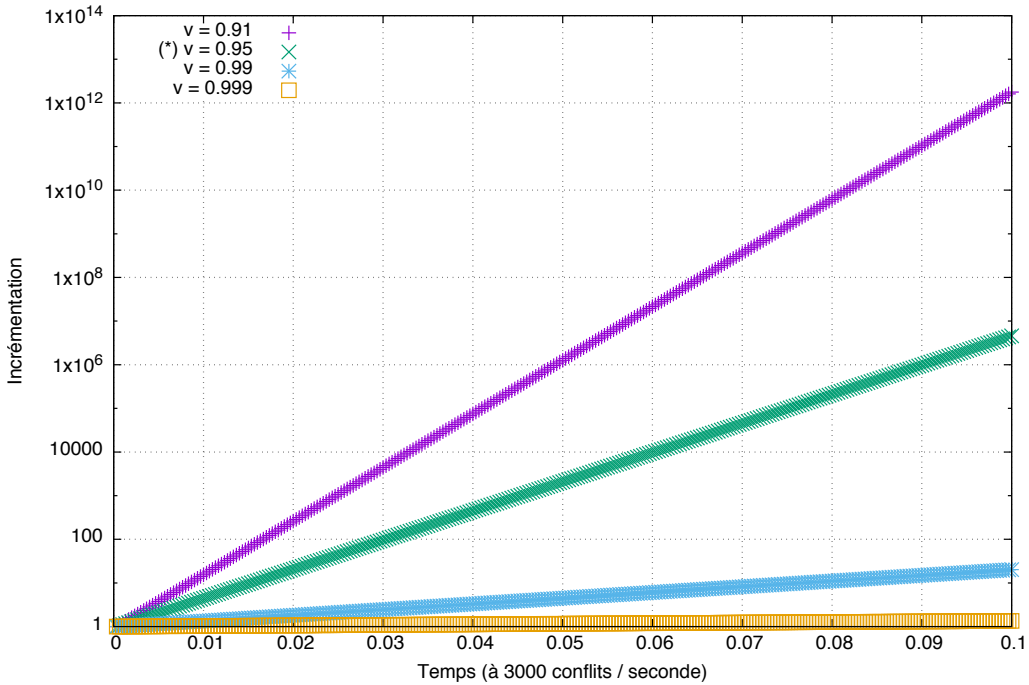
VSIDS : Variable State Independent Decaying Sum [Moskewicz2001]

Variable State Independent Decaying Sum

- Bump variables occuring in learnt clauses
- Divided scores by 2 every 256 conflicts : Favorize recent learnt clauses

EVSIDS : Exponential VSIDS [EEn03]

- Bump variables occuring in conflict analysis by b and change $b = v \times b$
- $v = 1.05$ by default ; $v = 1$: the past is not forgetten ; $v = 1.09$: Forget the past a lot

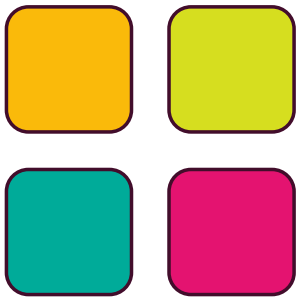


- The heuristics is highly dynamic....
- A good variable can becomes bad very quickly

Phase saving [Pipatsrisawat2007]

- Once the variable is selected, which polarity has to be chosen ?
- Simply keep track of the last used polarity. And replay it.
- Work very well with fast restarts
- Do not forget already solutions of sub-formula

- Take a look at a CDCL solver (the essentials of Minisat in fact) : Activity, decision procedure



Quality of Learnt Clauses

Glucose SAT solver [Audemard09]

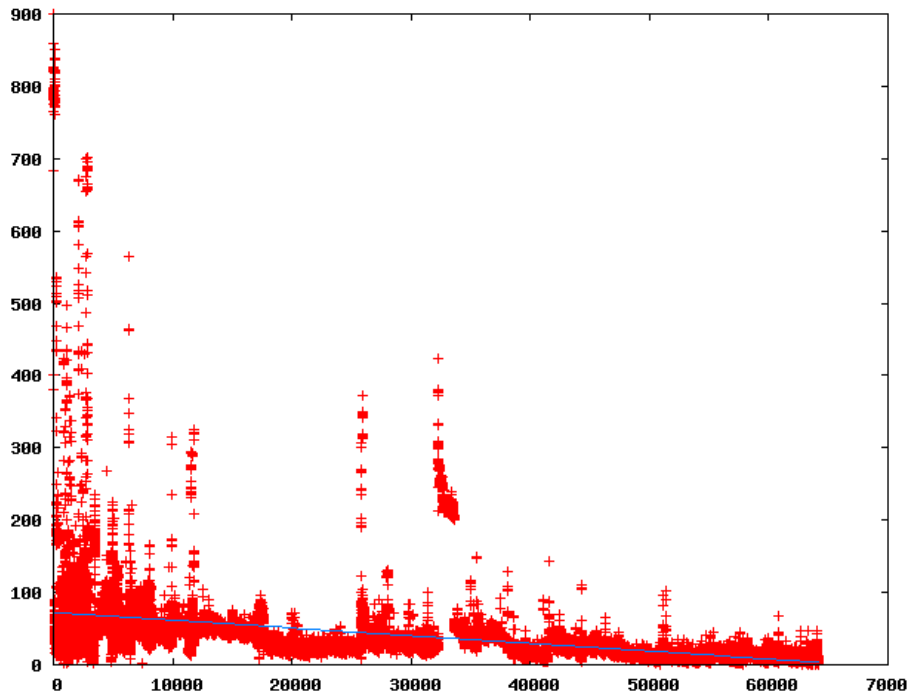
Intensive experiments is in the heart of GLUCOSE

- SAT solver developed by Laurent Simon (LABRI) and myself
- Based on Minisat !
- Essentially based on a static measure identifying good learnt clauses (LBD)
  - Agressive cleaning strategy : bad clauses have big LBD value
  - Dynamic restarts based on LBD
  - Other features...
- One of the SOTA solver since 2009
- A special hack glucose track since SAT'16

Main Features

- UNSAT proof generation (Thanks to Marijn)
- Incremental mode
- Parallel (multi-threads) mode
- New : a distributed version

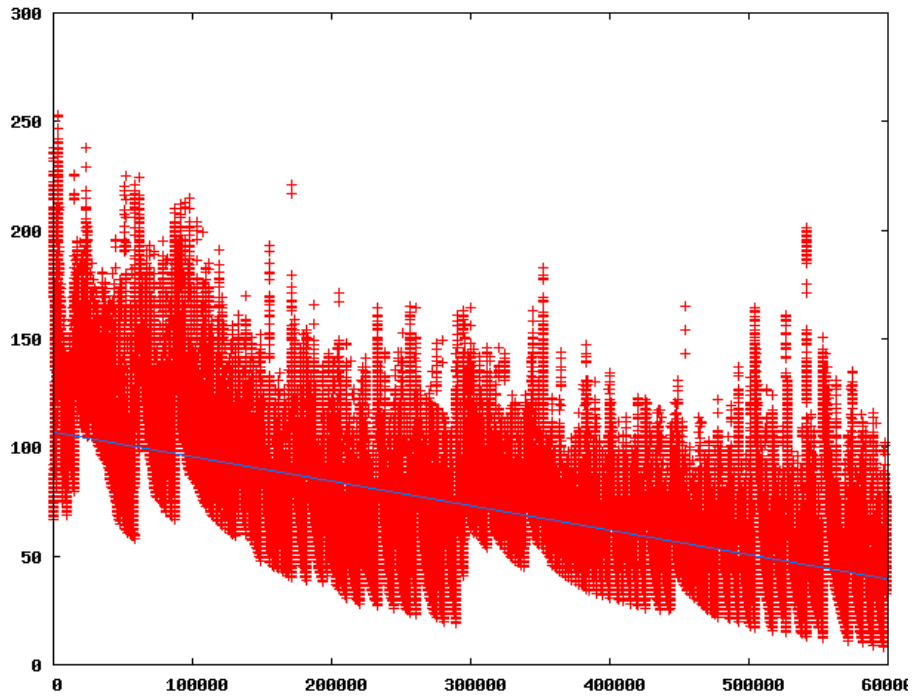
Many experiments behind Glucose . . .



een-pico-prop05-50 – UNSAT – 13,000 vars and 65,000 clauses

- For each conflict, we store the decision level where it occurs
- We also compute the linear regression on these points
- Gives an idea of the global behavior of the computation

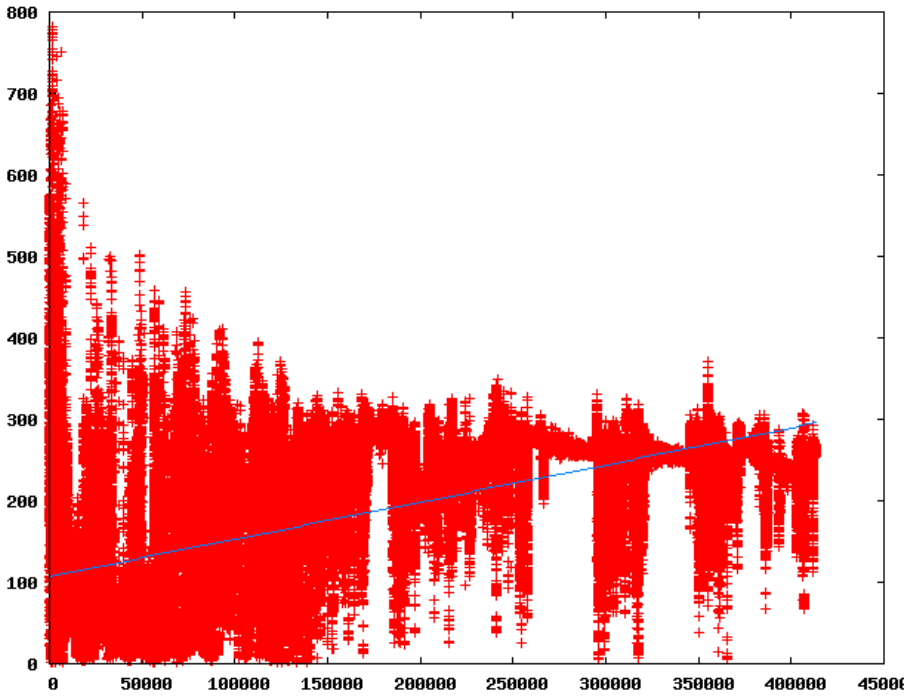
Many experiments behind Glucose ...



grieu-vmpc-s05-25 – SAT – 625 vars and 76,000 clauses

- For each conflict, we store the decision level where it occurs
- We also compute the linear regression on these points
- Gives an idea of the global behavior of the computation

Many experiments behind Glucose . . .



mizh-sha0-35-3 – SAT – 20,000 vars and 120,000 clauses

- For each conflict, we store the decision level where it occurs
- We also compute the linear regression on these points
- Gives an idea of the global behavior of the computation

Remarks

- Of course, we do not expect to feet curves
- We try to make observations of the behavior of a CDCL solver

AND...

Decreasing appears in a lot of problems

Series	#Benchs	% Decr.
een	8	62%
goldb	11	100%
grieu	7	71%
hoons	5	100%
ibm-2002	7	71%
ibm-2004	13	92%
manol-pipe	55	91%
miz	13	0%
schup	5	80%
simon	10	90%
vange	3	66%
velev	54	92%
all	199	83%

Intuitions

- A lot of dependencies between variables
  During search those variables will probably be propagated together inside blocks of propagations
- One needs to collapse independent blocks of propagated literals in order to reduce the decision level

The LBD score of a nogood is the number of different blocks of propagated literals

Intuitions

- A lot of dependencies between variables
  During search those variables will probably be propagated together inside blocks of propagations
- One needs to collapse independent blocks of propagated literals in order to reduce the decision level

The LBD score of a nogood is the number of different blocks of propagated literals

- LBD=2
  - Only one literal from the last decision level (the assertive one)
  - This literal will be **glued** to the other block
  - binary clauses have LBD equal to 2
- VSIDS + progress saving : this should occurs a lot ! ! !

Good clauses are GLUE clauses

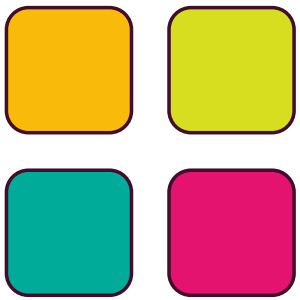
Managing learnt clauses

- Too many clauses and the BCP gets slower
- Not all learnt clauses are useful
- Remove many clauses
- Instead of removing clauses when needed, we eagerly remove them very often
- We can remove 95% of the clauses and get better !

Use the LBD measure

- Minisat uses a criteria based on the activity : a good clause in the past will be good in the future

- Take a look at a CDCL solver (the essentials of Minisat in fact) : ReduceDB



Restarts

Restarts : Introduction

- Very important component
- Initially introduced to avoid heavy tail distribution on local search algorithms
- With no restart, the choice of the first decision variables is crucial !
- Unassign all variables : Keep the dynamic heuristics ! !

Clause learning + Phase saving + Learning

- The solver directly goes to the same search space, but with a distinct path
- When to restart ?
- Relationship with learning (selecting a variable on the top of the search tree means that will probably dont use it during resolution steps)

Static restart schemes

- Geometric, introduced in Minisat 2.0 : $100 * 1.5^{nb_{restarts}}$

100 150 225 337 506 759 1139 1708 2562 3844 5766 8649 12973 19460

- Luby restarts : (1,2,1,2,4,1,2,1,2,4,8,1,2,1,2,4,1,2,1,2,4,8,16,...). Multiplied by a constant
  - it is exponentially increasing, but exponentially slowly, thus limiting the risk of searching for a long time in the wrong search space
  - Lubys strategy is optimal...When we blindly exploite/explore a process !
  - Luby with constant 6 seems optimal and efficient if the trail is reused

100 200 100 200 400 100 200 100 200 400 800 100 ...

- Inner-Outer :
  - 2 geometric sequences : inner and outer.
  - Restarts every inner conflicts, if inner $>$ outer, increase outer and set inner to the initial value
  - ensure that restarts were guaranteed to increase and that fast restarts occurred more often than the geometric series

100 100 110 100 110 121 100 110 121 133 100 110 121 133

Dynamic restart schemes

When the solver does not make any progress, restart

- Agility : based on the polarity of the phase saving mechanism
  - If most of the variables are forced against their saved polarity, then the restart is postponed : the solver might find a refutation soon
  - If polarities are stalling, the scheduled restart is triggered

- Width-based scheme
  - Each time the learnt clause has a size greater or equal to a threshold, a penalty is set
  - After a given number of penalties, a restart is triggered

Glucose restarts [Audemard12]

Dynamic scheme based on LBD

- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

Glucose restarts [Audemard12]

Dynamic scheme based on LBD

- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

```
// In case of conflict
... compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

Glucose restarts [Audemard12]

Dynamic scheme based on LBD

- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

```
// In case of conflict
... compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

- Perform at least X conflicts before restarting

Glucose restarts [Audemard12]

Dynamic scheme based on LBD

- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

```
// In case of conflict
... compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

- Perform at least X conflicts before restarting
- Average over last X LBD become too big wrt total average

Glucose restarts [Audemard12]

Dynamic scheme based on LBD

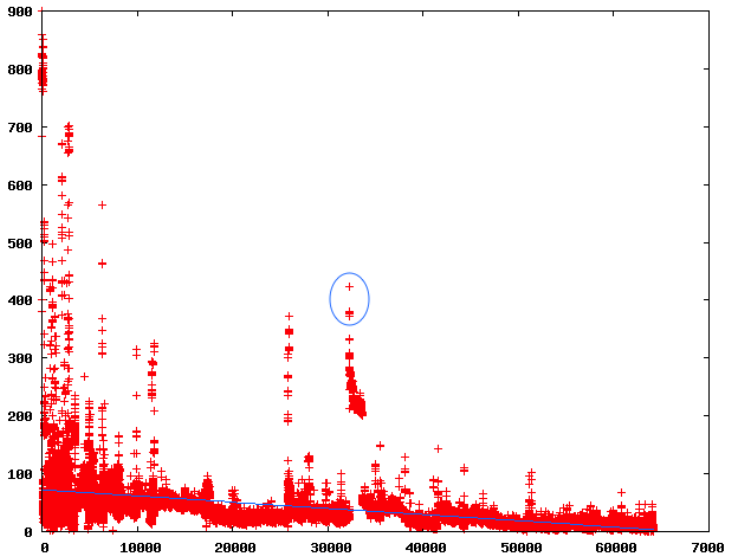
- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

```
// In case of conflict
... compute learnt clause c;
sumLBD+=c.lbd();
queueLBD.push(c.lbd());
if(queueLBD.isFull() && queueLBD.avg()*K>sumLBD/nbConflicts) {
    queueLBD.clear();
    restart();
}
```

- GLUCOSE 1.0 and 2.0 : X=100 and K=0.7
- GLUEMINISAT and glucose 2.1 : X=50 and K=0.8

Targeting SAT too

- Frequent restarts seems not very good in case of SAT instances
- Agressive restarts : some global assignments can be dropped ! !



een-pico-prop05-50 – UNSAT – 13,000 vars and 65,000 clauses

- The number of decisions before reaching a conflict suddenly increases
- It seems that the solver goes trough a difficult part of the search space, and is closed to a full assignment.
- We can safely suppose that the solver needs to stay on this search space in order to find a solution

Targeting SAT too

Delay restarts if total of assignments suddenly increases

- We compare the current trail size with the average of the last Y ones
- We use a bounded queue of the last Y trail size when reaching a conflict (`queueTrail`)

Targeting SAT too

Delay restarts if total of assignments suddenly increases

- We compare the current trail size with the average of the last Y ones
- We use a bounded queue of the last Y trail size when reaching a conflict (`queueTrail`)

```
// In case of conflict
queueTrail.push(trail.size());
if(queueTrail.isFull() && trail.size()>T*queueTrail.avg()) {
   queueLBD.clear();
}

compute learnt clause c
 ...
```

Targeting SAT too

Delay restarts if total of assignments suddenly increases

- We compare the current trail size with the average of the last Y ones
- We use a bounded queue of the last Y trail size when reaching a conflict (`queueTrail`)

```
// In case of conflict
queueTrail.push(trail.size());
if(queueTrail.isFull() && trail.size()>T*queueTrail.avg()) {
   queueLBD.clear();
}

compute learnt clause c
 ...
```

- The total number of assignments suddenly increases

Targeting SAT too

Delay restarts if total of assignments suddenly increases

- We compare the current trail size with the average of the last Y ones
- We use a bounded queue of the last Y trail size when reaching a conflict (`queueTrail`)

```
// In case of conflict
queueTrail.push(trail.size());
if(queueTrail.isFull() && trail.size()>T*queueTrail.avg()) {
   queueLBD.clear();
}

compute learnt clause c
 ...
```

- The total number of assignments suddenly increases
- Postpone restart

Targeting SAT too

Delay restarts if total of assignments suddenly increases

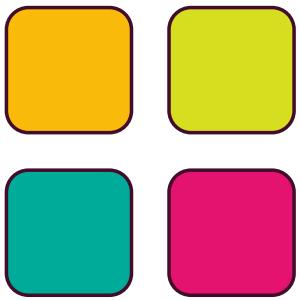
- We compare the current trail size with the average of the last Y ones
- We use a bounded queue of the last Y trail size when reaching a conflict (`queueTrail`)

```
// In case of conflict
queueTrail.push(trail.size());
if(queueTrail.isFull() && trail.size()>T*queueTrail.avg()) {
   queueLBD.clear();
}

compute learnt clause c
 ...
```

- The total number of assignments suddenly increases
- Postpone restart
- $Y = 5000$ and $T = 1.4$ appears to be good

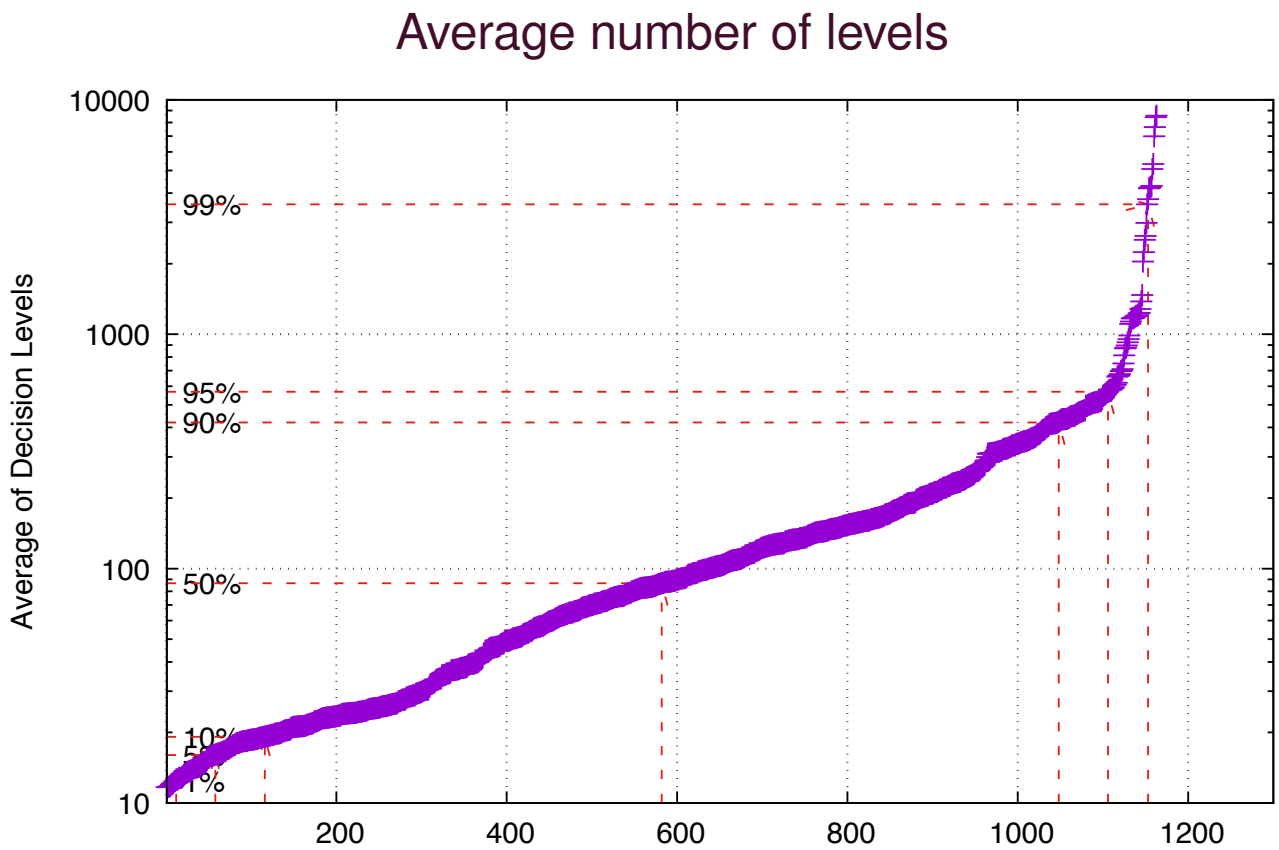
- Take a look at a CDCL solver (the essentials of Minisat in fact) : another branch



Behaviour of CDCL solvers

Outliers everywhere !

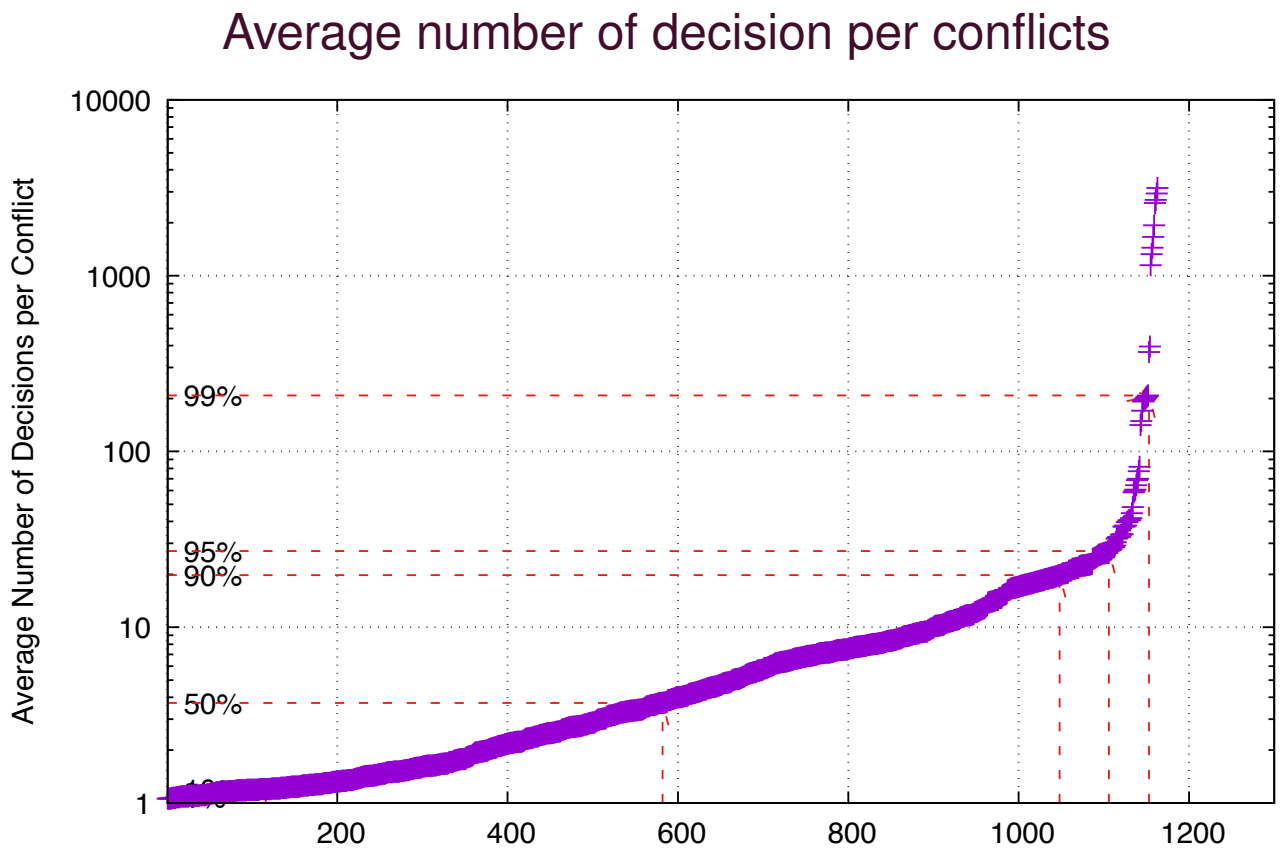
We try to understand CDCL solvers but all problems are distincts !



stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

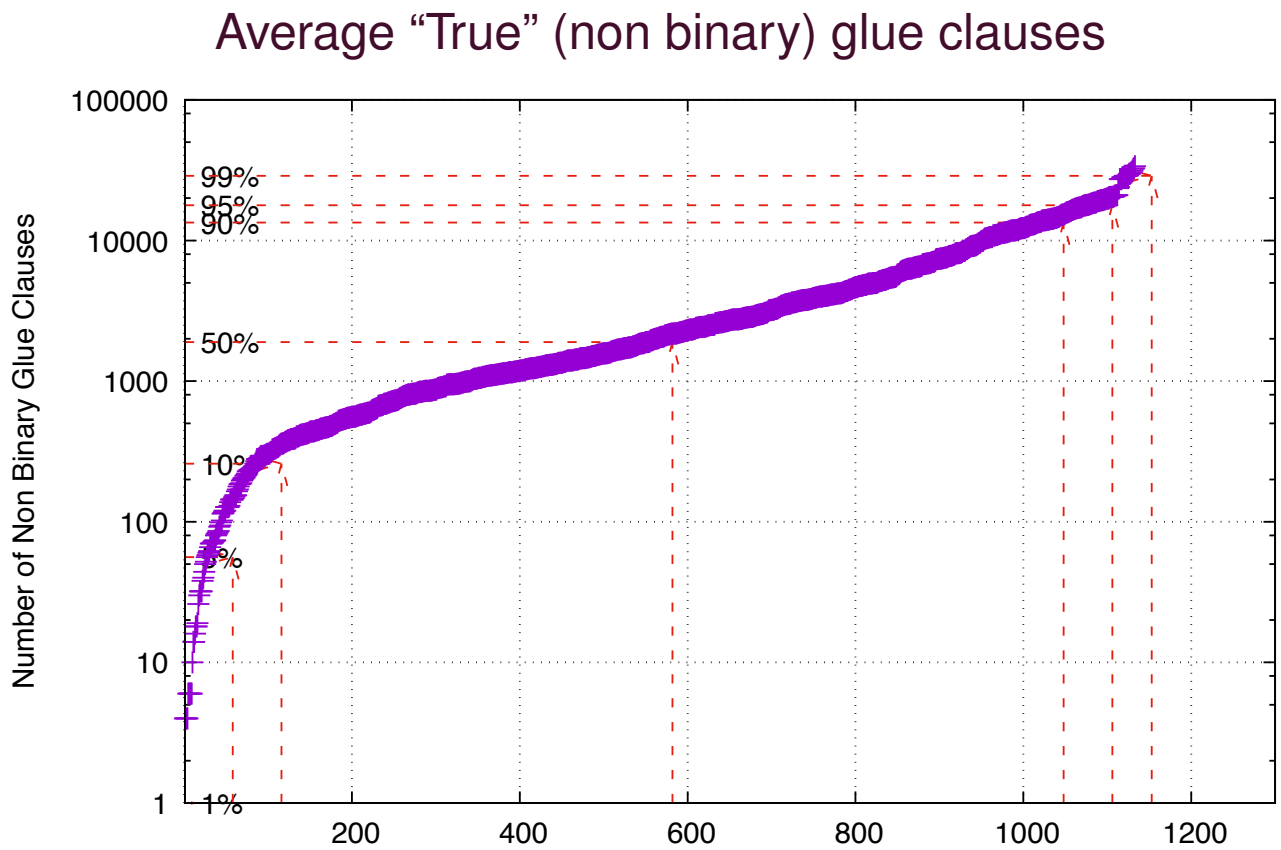
We try to understand CDCL solvers but all problems are distincts !



stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

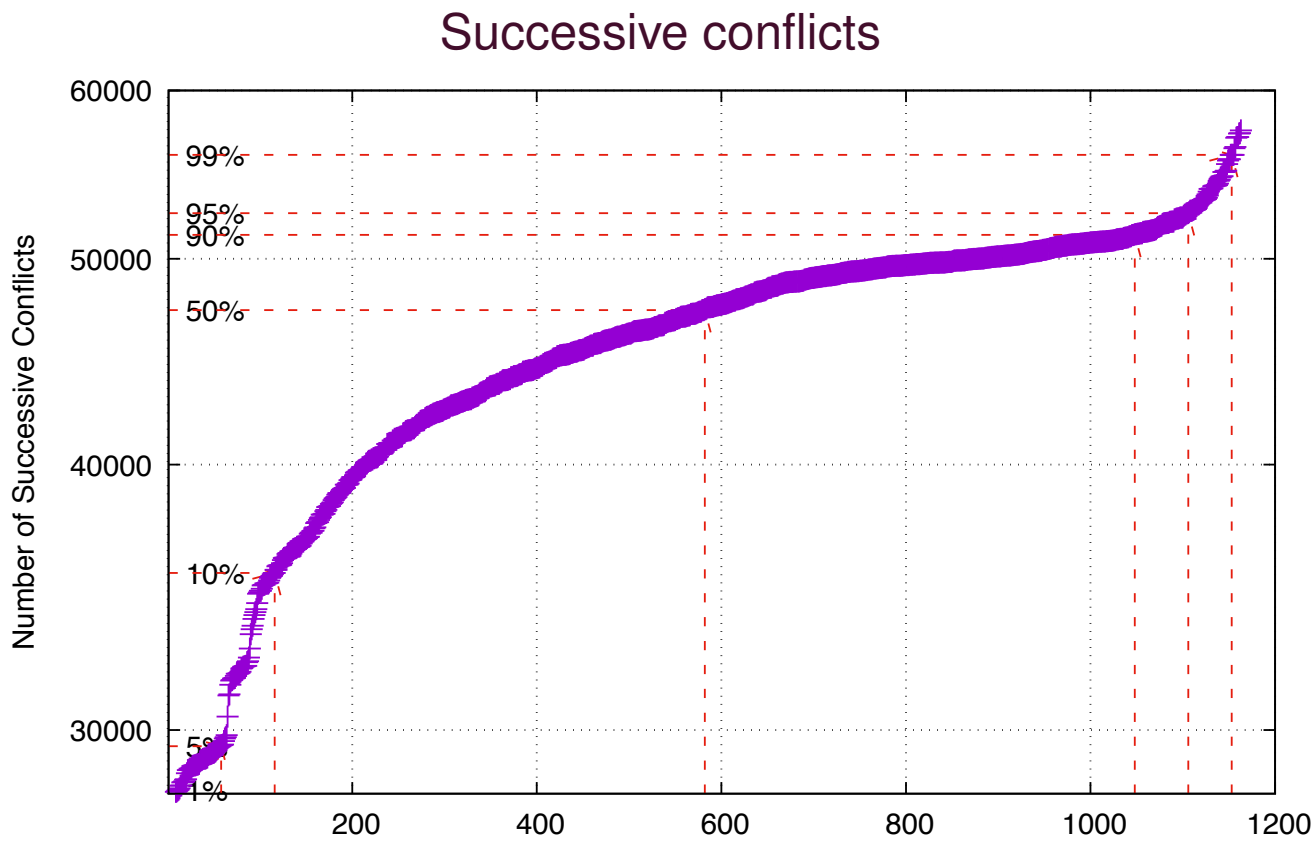
We try to understand CDCL solvers but all problems are distincts !



stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

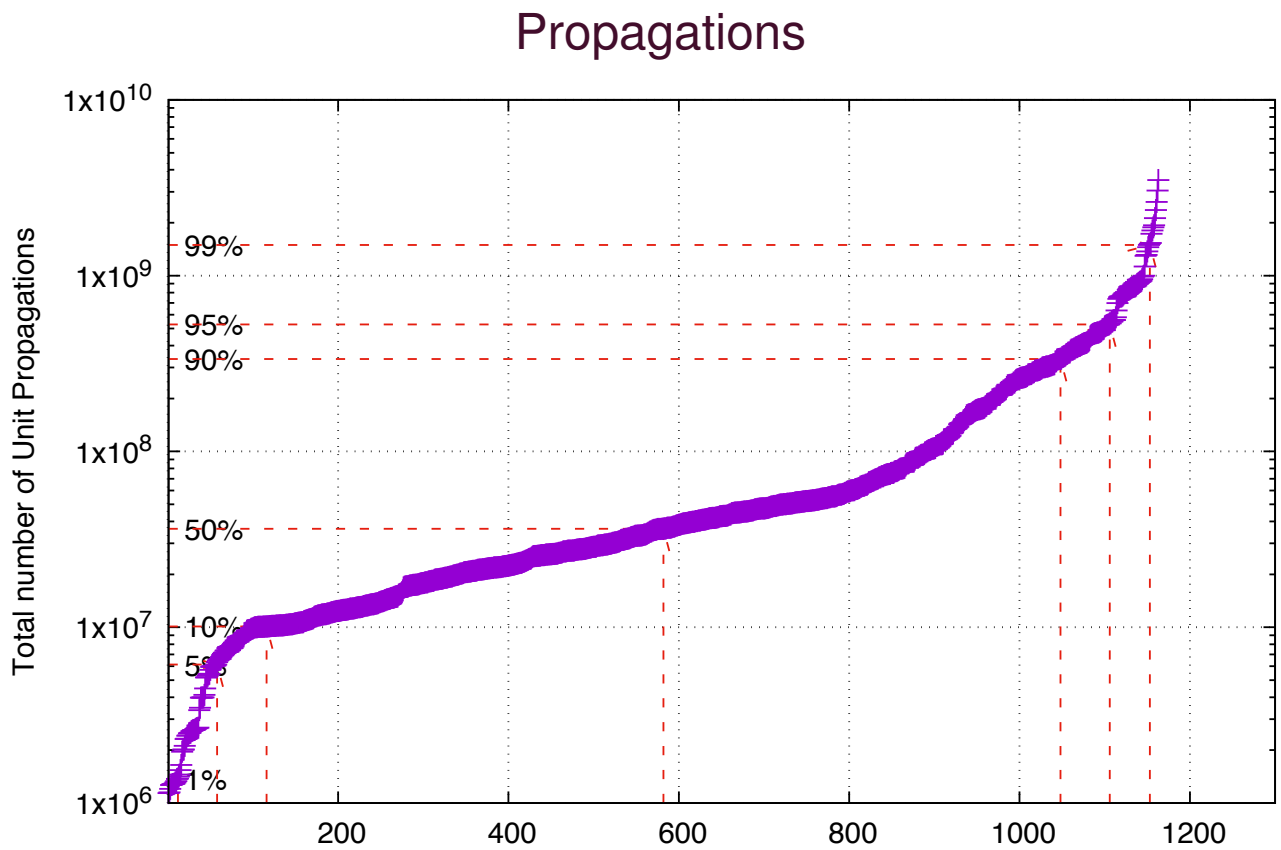
We try to understand CDCL solvers but all problems are distincts !



stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

We try to understand CDCL solvers but all problems are distincts !

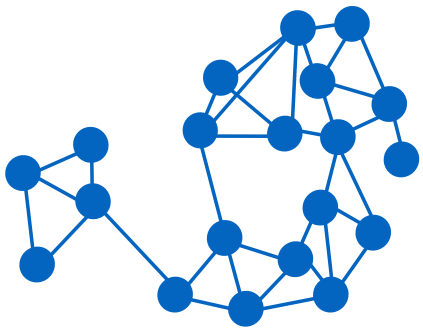


stats after 10,000 conflicts on 1164 “not easy” problems from all previous contests

Try to Explain Efficiency of CDCL Solvers

Instances and Community Structures

- Create a graph representation of a SAT instance :
  One node per variable
  two nodes are linked if associated variables appear in same clause
- Partition nodes
- Many connections inside a group
- Few connections outside a group

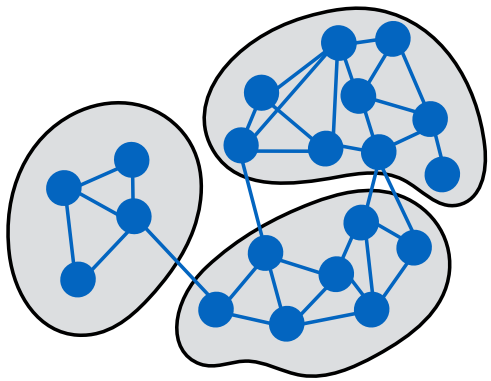


- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

Try to Explain Efficiency of CDCL Solvers

Instances and Community Structures

- Create a graph representation of a SAT instance :
  One node per variable
  two nodes are linked if associated variables appear in same clause
- Partition nodes
- Many connections inside a group
- Few connections outside a group

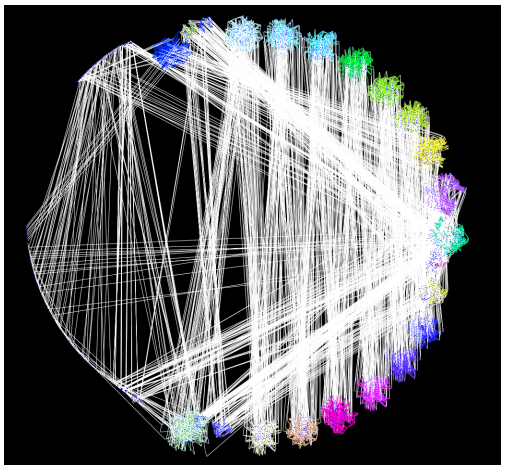


- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

Try to Explain Efficiency of CDCL Solvers

Instances and Community Structures

- Create a graph representation of a SAT instance :
  One node per variable
  two nodes are linked if associated variables appear in same clause
- Partition nodes
- Many connections inside a group
- Few connections outside a group



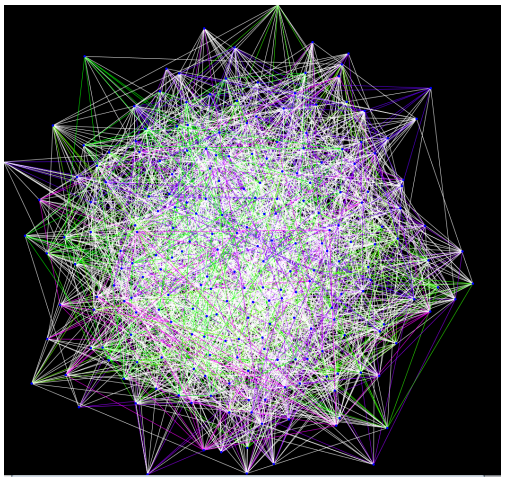
BMC instance

- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

Try to Explain Efficiency of CDCL Solvers

Instances and Community Structures

- Create a graph representation of a SAT instance :
  One node per variable
  two nodes are linked if associated variables appear in same clause
- Partition nodes
- Many connections inside a group
- Few connections outside a group



Random instance

- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

Try to Explain Efficiency of CDCL Solvers

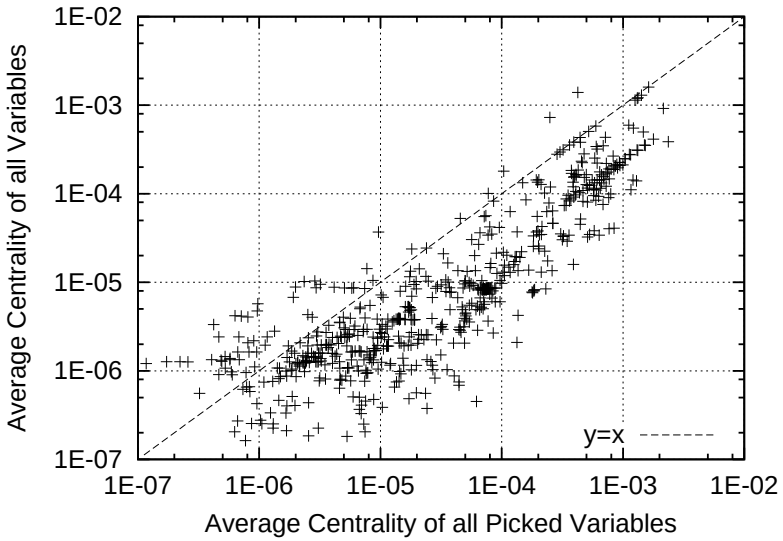
Centrality of variables

- Here again, a graph representation of a SAT instance
- Measure the centrality of variables
- Establish a link between
  - Centrality and VSIDS heuristic
  - Centrality and learning

Try to Explain Efficiency of CDCL Solvers

Centrality of variables

- Here again, a graph representation of a SAT instance
- Measure the centrality of variables
- Establish a link between
  - Centrality and VSIDS heuristic
  - Centrality and learning



- Decision variables are central !

Try to Explain Efficiency of CDCL Solvers

Centrality of variables

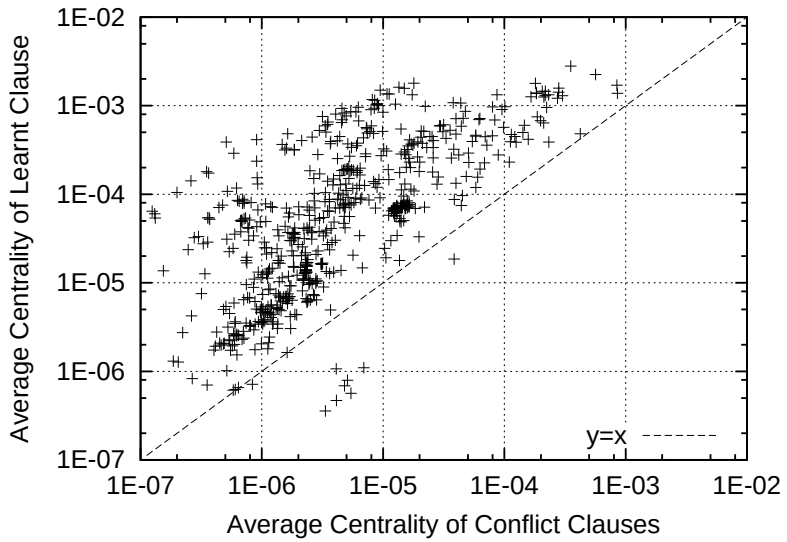
- Here again, a graph representation of a SAT instance
- Measure the centrality of variables
- Establish a link between
  - Centrality and VSIDS heuristic
  - Centrality and learning

- Decision variables are central !

Try to Explain Efficiency of CDCL Solvers

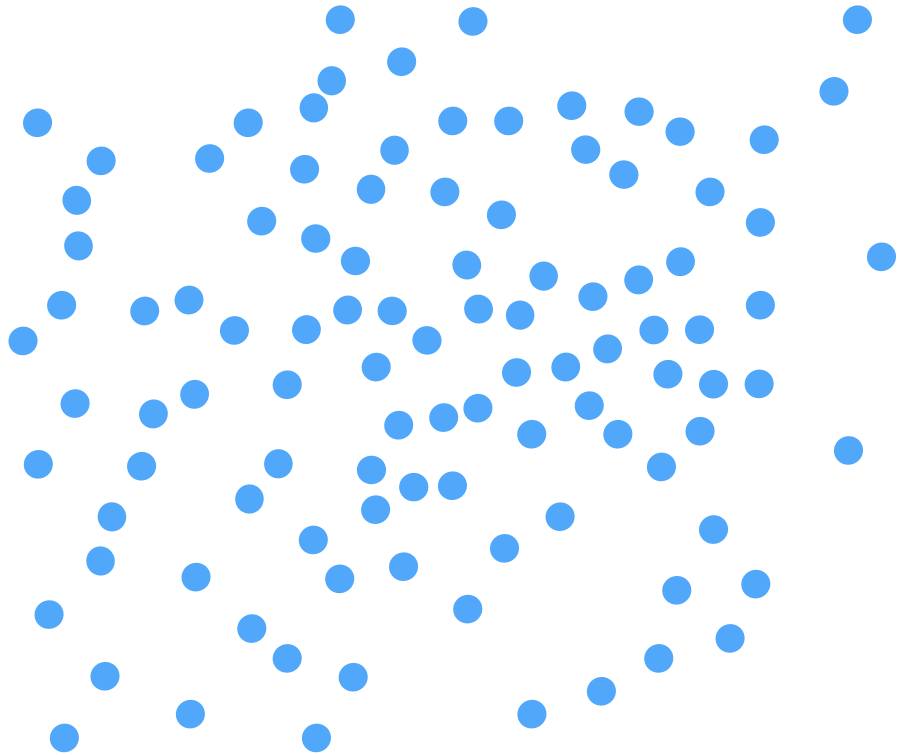
Centrality of variables

- Here again, a graph representation of a SAT instance
- Measure the centrality of variables
- Establish a link between
  - Centrality and VSIDS heuristic
  - Centrality and learning



- Decision variables are central !
- Learnt clauses are central

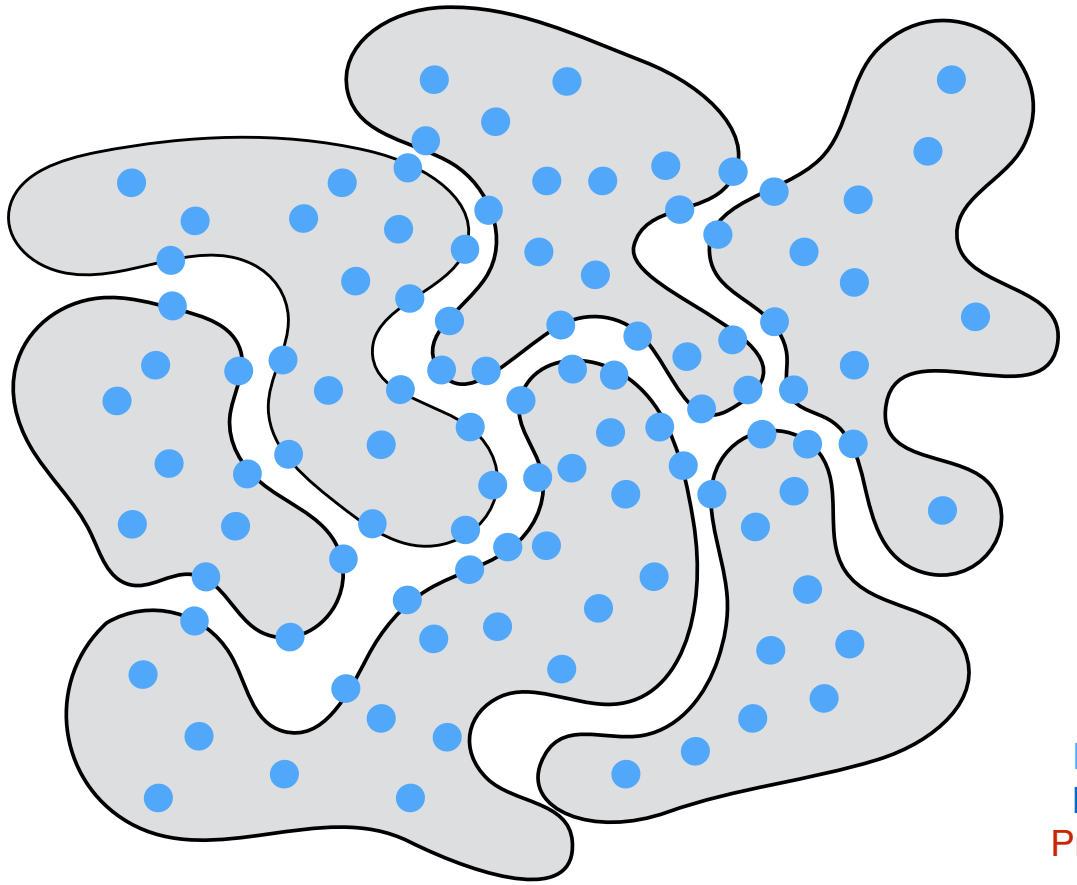
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

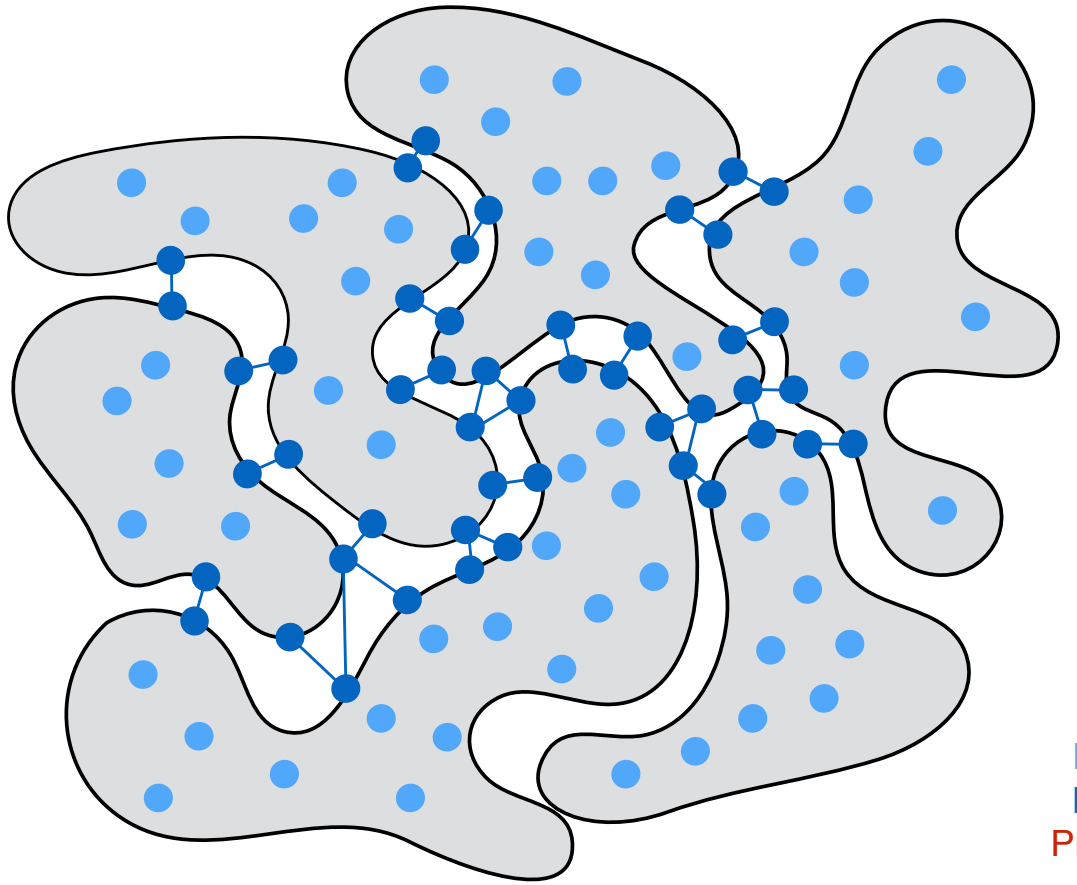
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

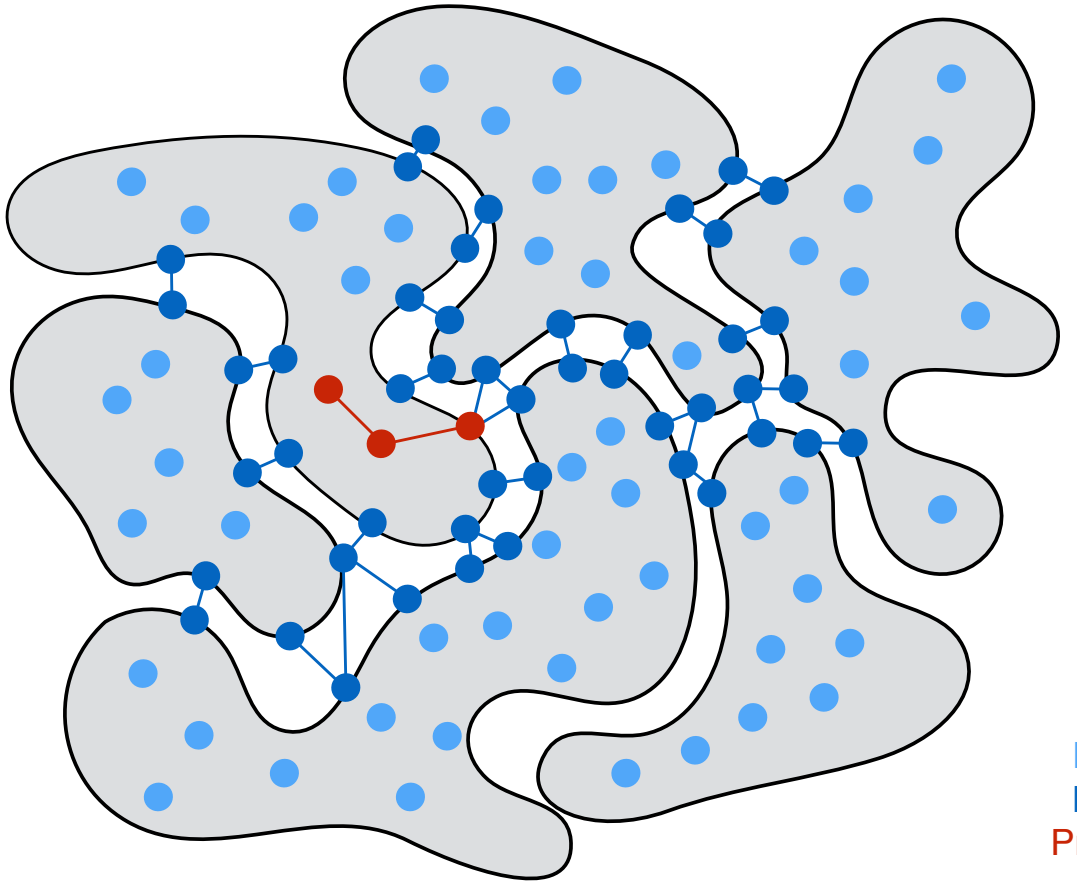
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

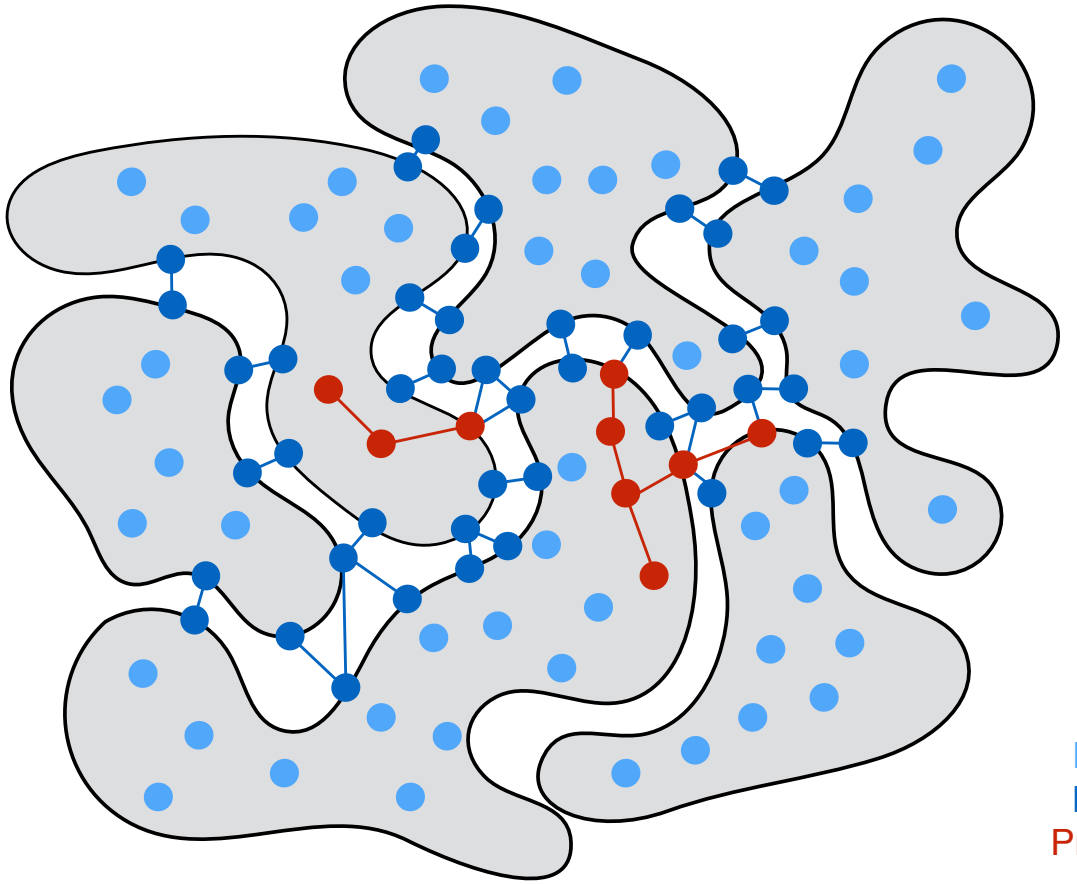
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

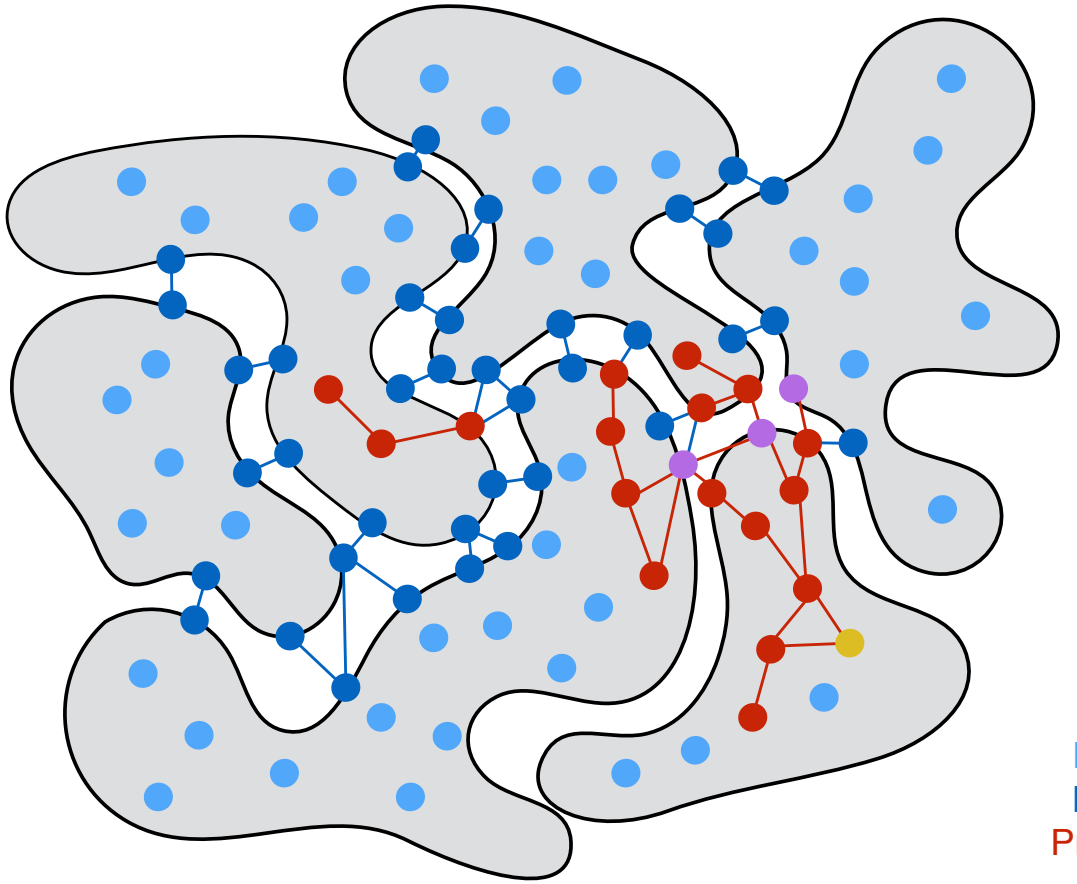
© Laurent Simon

A Possible Illustration of CDCL Solvers



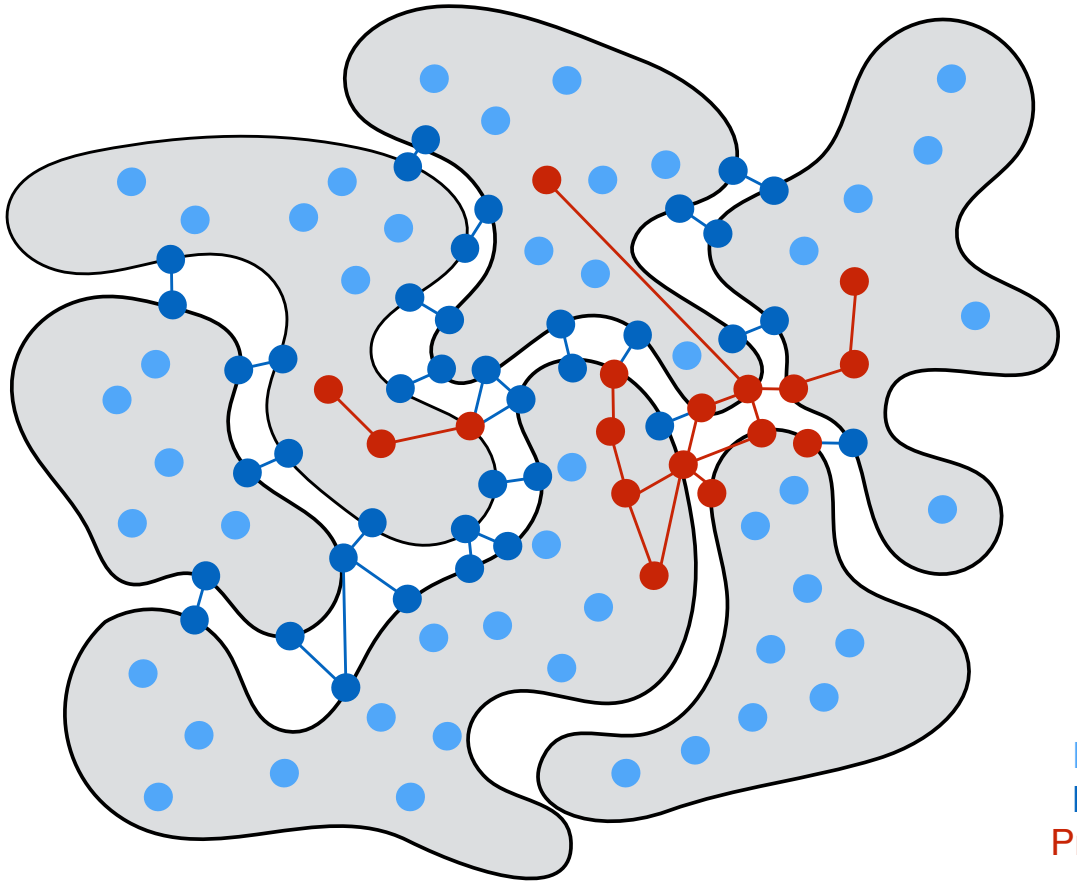
© Laurent Simon

A Possible Illustration of CDCL Solvers



© Laurent Simon

A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

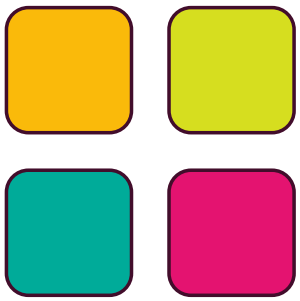
Many Questions

One knows how to write an efficient solver
Harder to explain the good performances

- CDCL is not DPLL : frequent restarts, deletion of many clauses
- Look for a model or a contradiction ?
- Learning can be bad
- Restarts are not restarts but provide different paths
- Try to help the solver is (frequently) a bad idea

Performances can always be improved

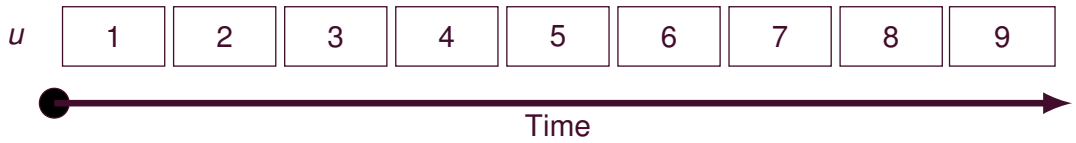
- On UNSAT instances :
  - 50% of learnt clauses are useless for the proof
  - 20% of propagations are useful
- But, just few papers these last years with novel and powerfull SAT techniques



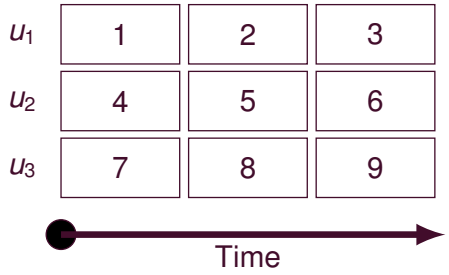
Parallel solvers

Issues related to parallelism

Sequential algorithm



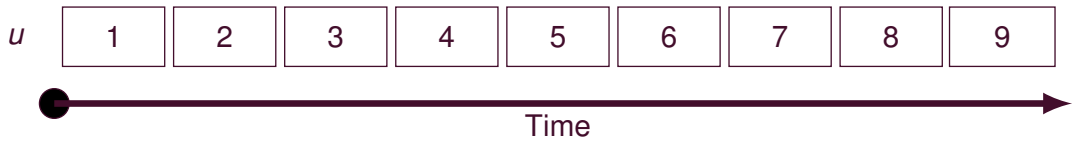
Parallel algorithm



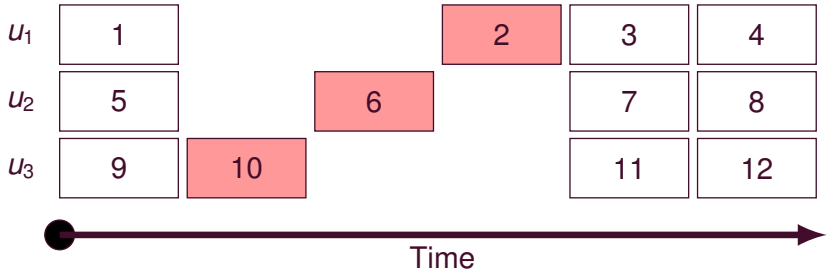
Division of the global tasks in subtasks

Issues related to parallelism

Sequential algorithms

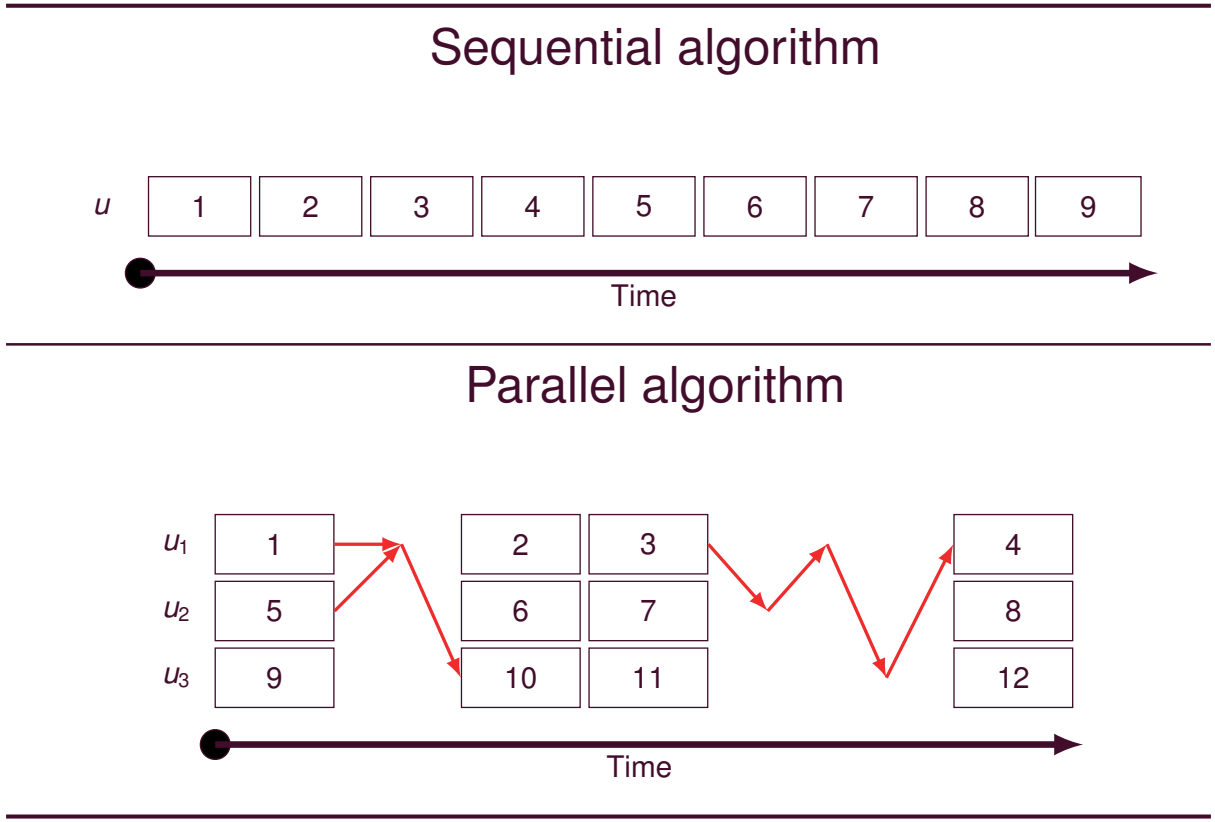


Parallel Algorithm



Dependency problems

Issues related to parallelism

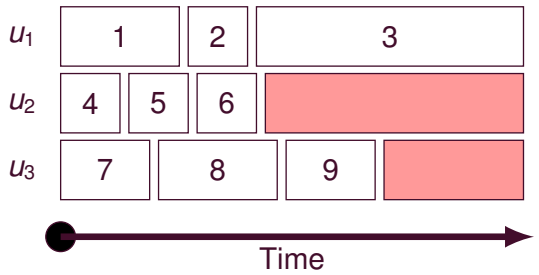


Slowdown due to communications

Issues related to parallelism

Sequential algorithm

Parallel algorithm

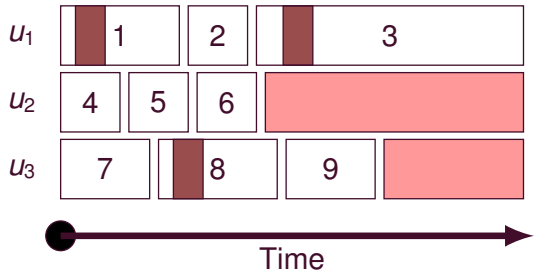


Load balancing

Issues related to parallelism

Sequential algorithm

Parallel algorithm

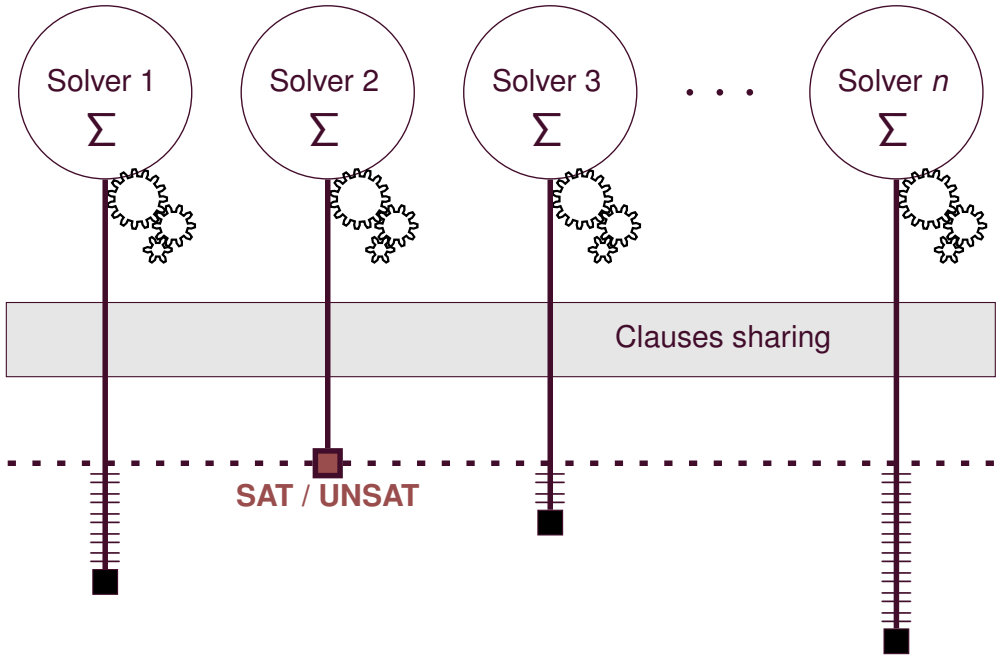


Redundant work

Solving SAT in parallel

- Not so easy to parallelize SAT solvers (in a efficient way)
- Bad idea : parallelisation of the BCP engine
- 2 main approaches : portfolio and divide & Conquer

Portfolio approach

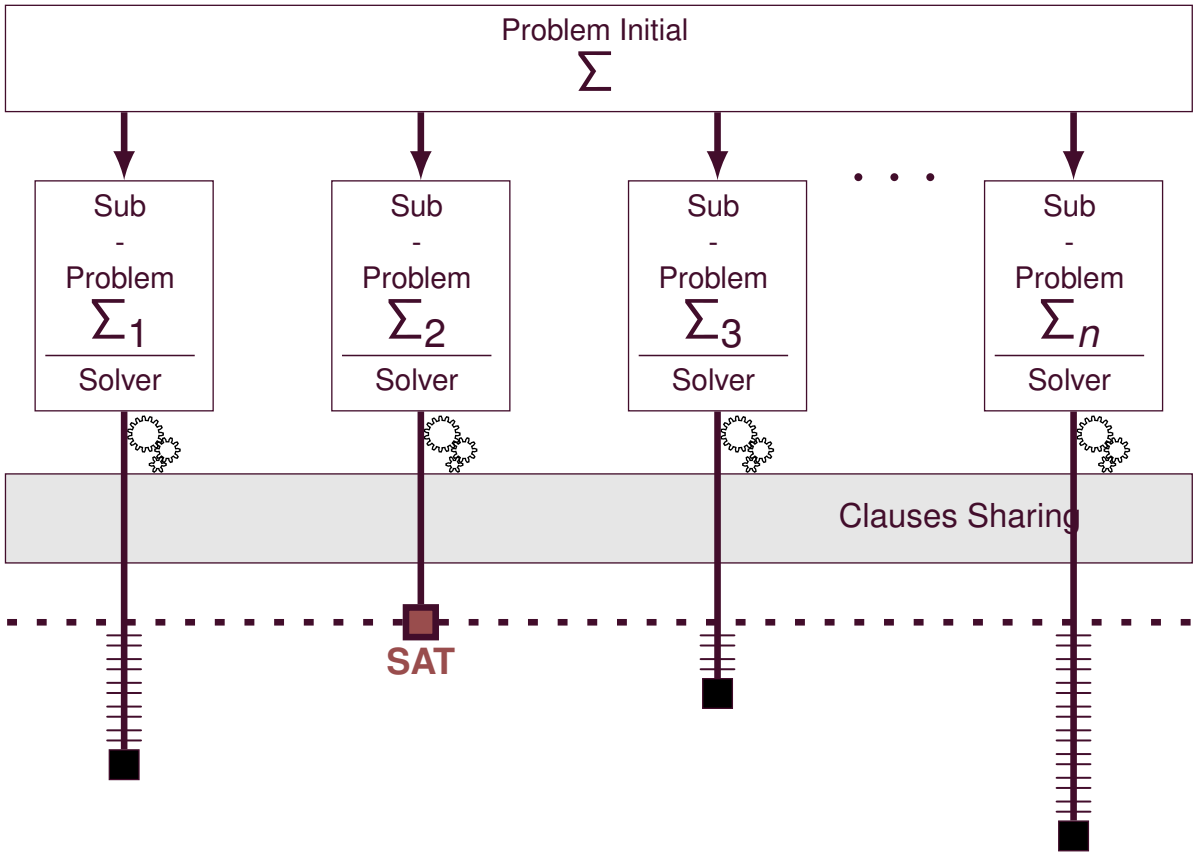


- Each solver works on the same (original) formula
- The first that successes is the winner
- Allow to easily share clauses.

Clauses sharing is essential !

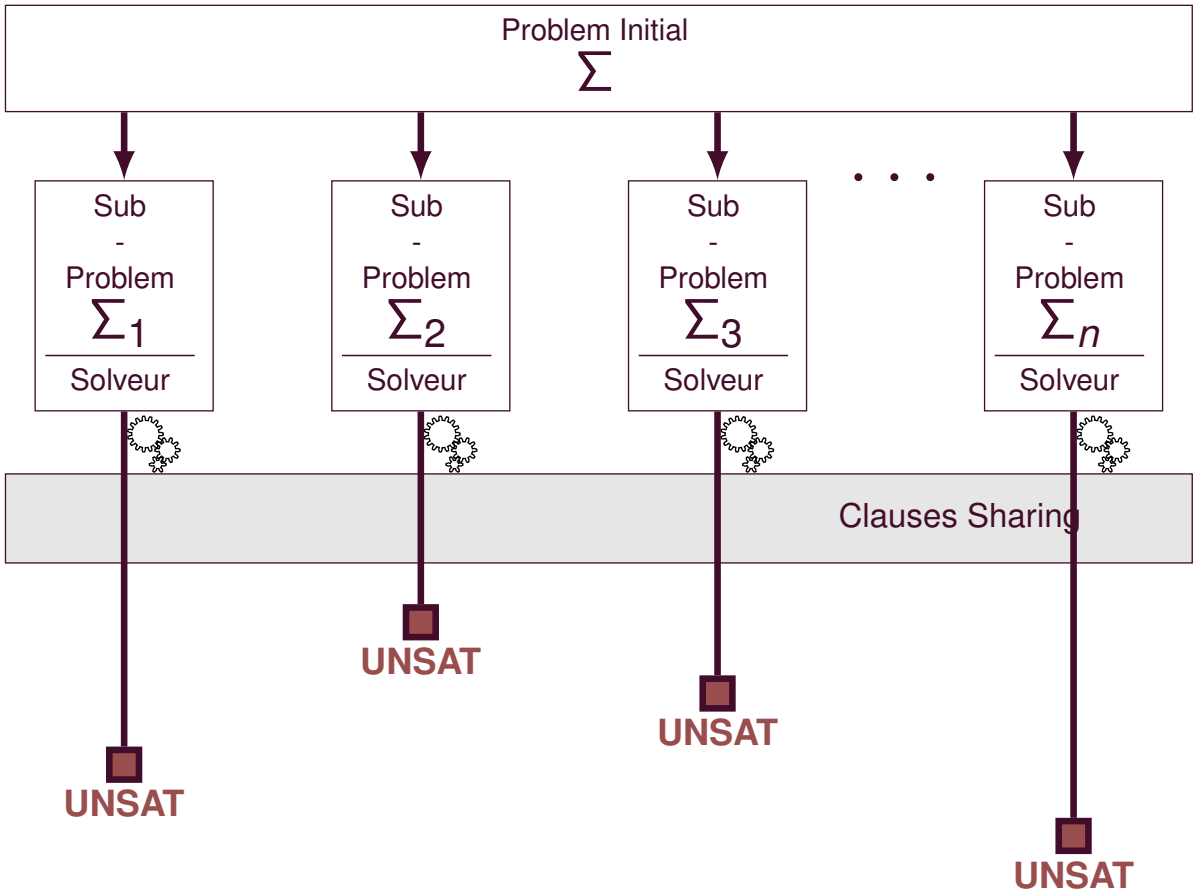
Divide and Conquer approach

- Split the search space
- Clauses sharing needs to use assumptions (see later)
- The end of the search depends on the status of the formula



Divide and Conquer approach

- Split the search space
- Clauses sharing needs to use assumptions (see later)
- The end of the search depends on the status of the formula



Syup : Many glucose in parallel

- syrup is the parallel version of glucose
- Portfolio solver (each one is a glucose)
- Dedicated stategy to import and export clauses

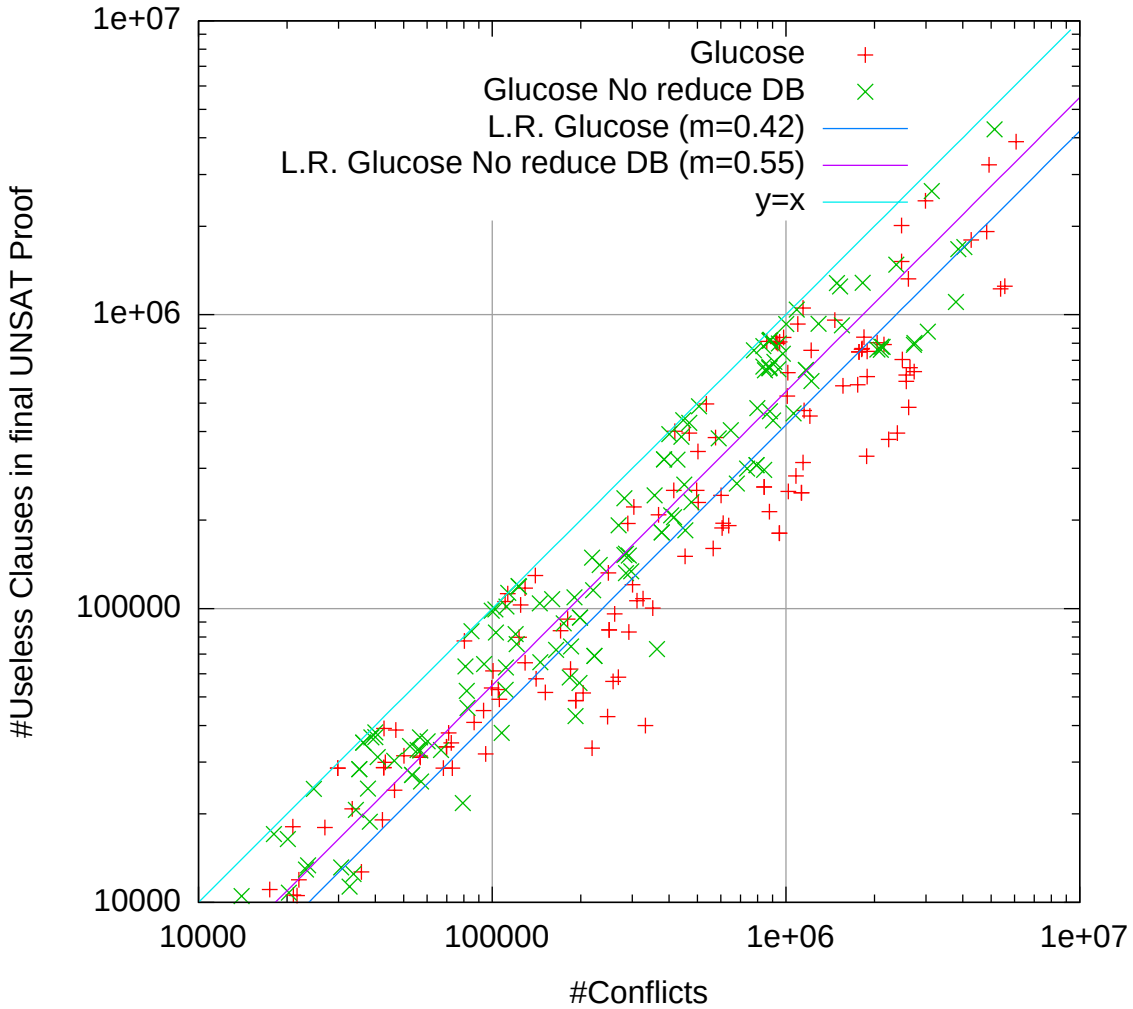
Many Cores, many more problems

Sharing clauses in parallel has many drawbacks

- Imported clauses can be bad (noise, wrong way, . . .)
- Imported clauses can be subsumed / useless
- Imported clauses can dominate learnt clauses
- Each thread has to manage many more clauses
- Many side effects on all core components

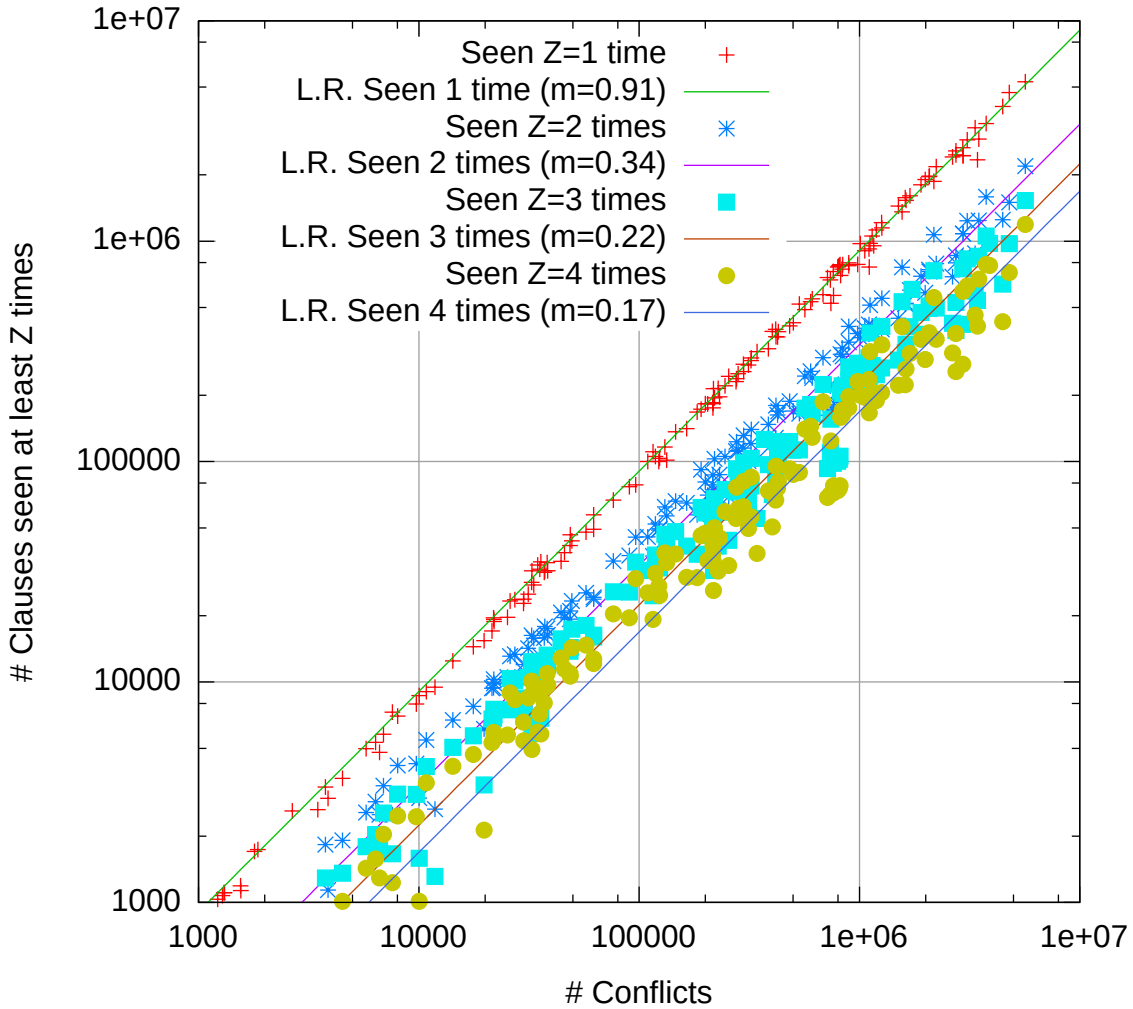
Parallel solvers limit the number of shared clauses (criterla : size, lbd...)

Many useless clauses even in single engine solvers



- x-axis : Number of conflicts
- y-axis : Useless clauses in final proof (UNSAT formulas)

How many times clauses are seen in conflicts



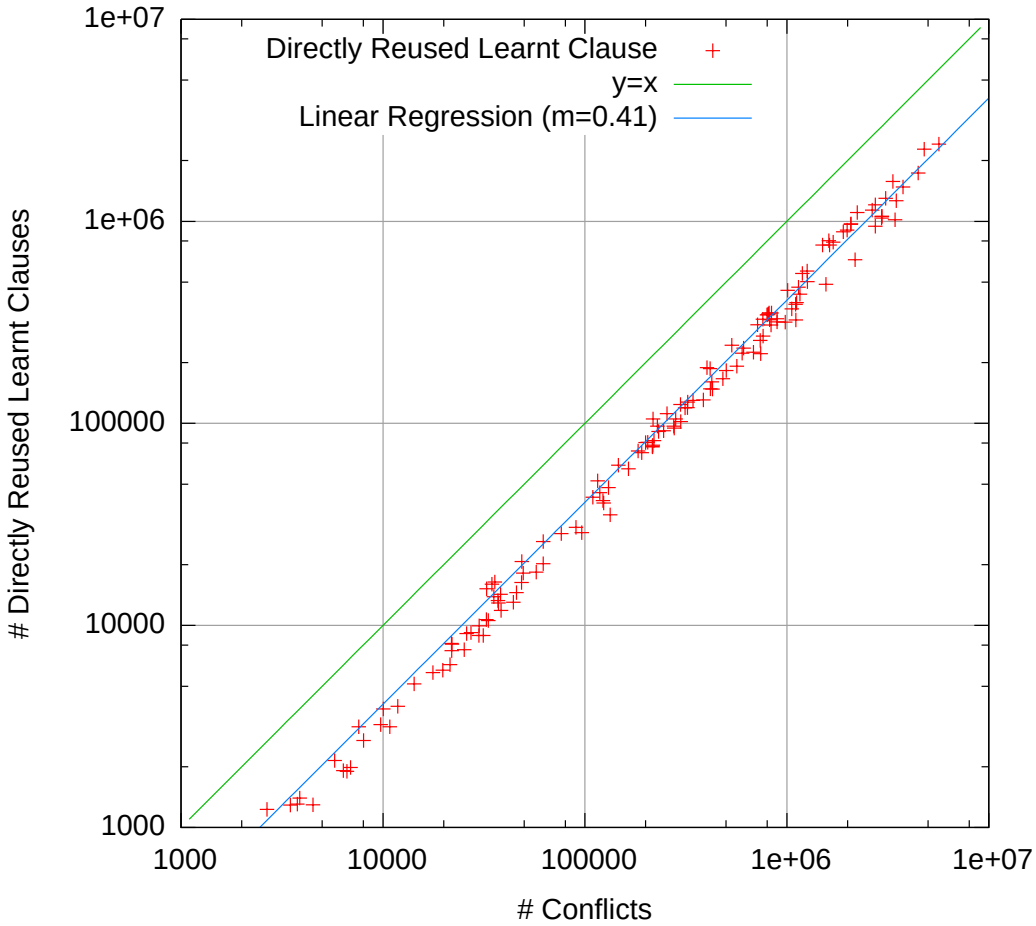
- x-axis : Number of conflicts
- y-axis : Number of clauses seen at least Z times

Outcomes

A lot of useless clauses even in a single engine !

- In parallel, this situation will be even worse !
- Why sending a clause that is even not locally interesting ?
- **We will consider clauses seen 2-times** (only 34% of learnt clauses)
  Already filter out the majority of clauses
- How to efficiently detect them ?
  Is there a window of recent clauses to check ?
  Can we check only recent learnt clauses (and save time) ?

Learnt and Directly Reused Clauses



- x-axis : Number of imported clauses (sum over 8 threads)
- y-axis : Number of promoted clauses (sum over 8 threads)

91% of clauses seen at least 1 time. Only 41% are immediately seen.

Lazy Exportation of Clauses

Clauses are sent during conflict analysis

- We only export clauses when seen 2 times in conflict analysis and :

- Clauses with LBD $\leq$ median(LBD) and size $\leq$ average(SIZE)

Limits updated at each clause database cleaning

Unary clauses and very glue clauses are immediately sent

Lazy because we wait to have a good chance of local interest before considering sending it

Lazy Importation of Clauses

Problem of clauses importation

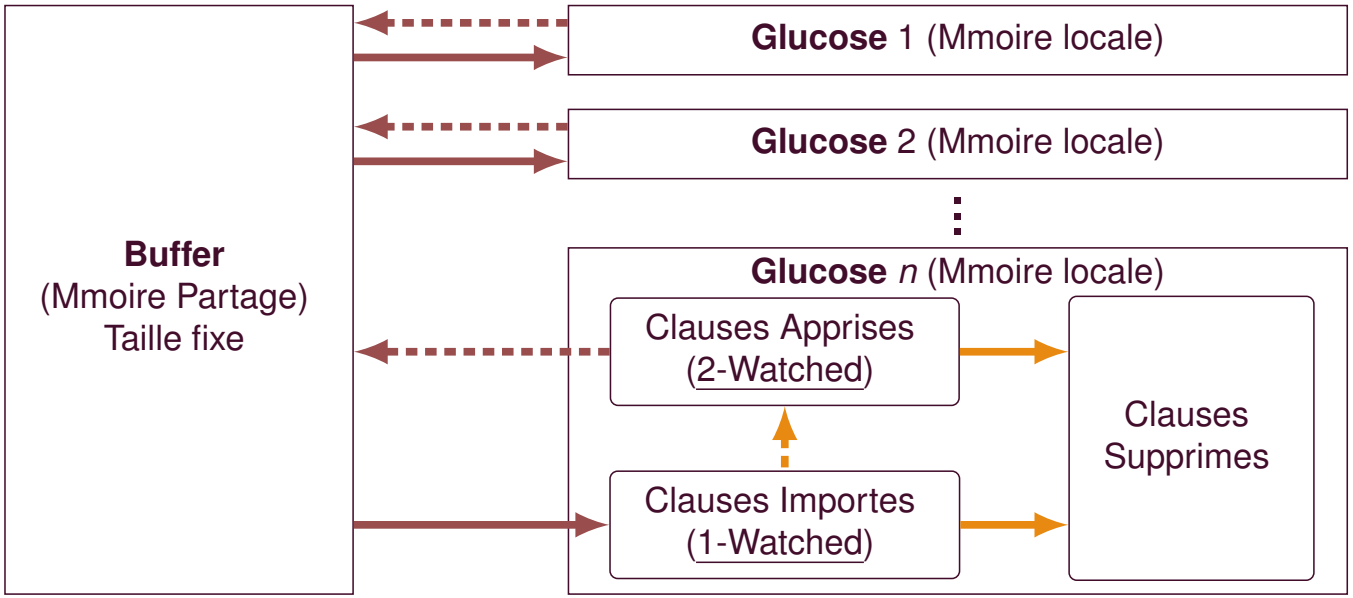
- can destroy the current search effort
- clauses can be redundant
- many clauses to manage (performance impact)

How to be sure an imported clause is interesting before considering it ?

Probation

Idea : put it in probation

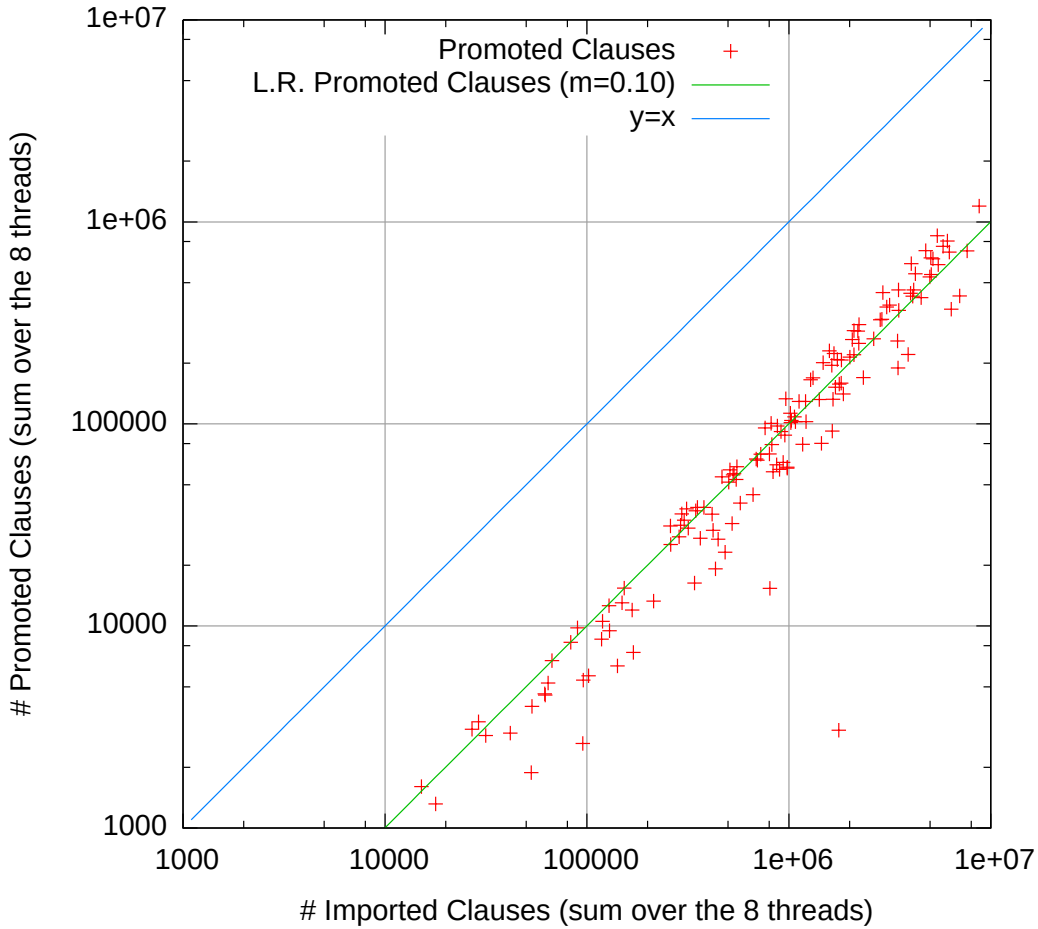
- Imported clauses are put in a 1-Watched scheme
- Will be promoted to a 2-Watched scheme only if found empty



Other Advantages

- Less efforts for propagations
- Can be seen as a dynamic Freezing/Reactivating strategy
- Clauses are imported with a local (and correct) LBD value

How many promotions ?

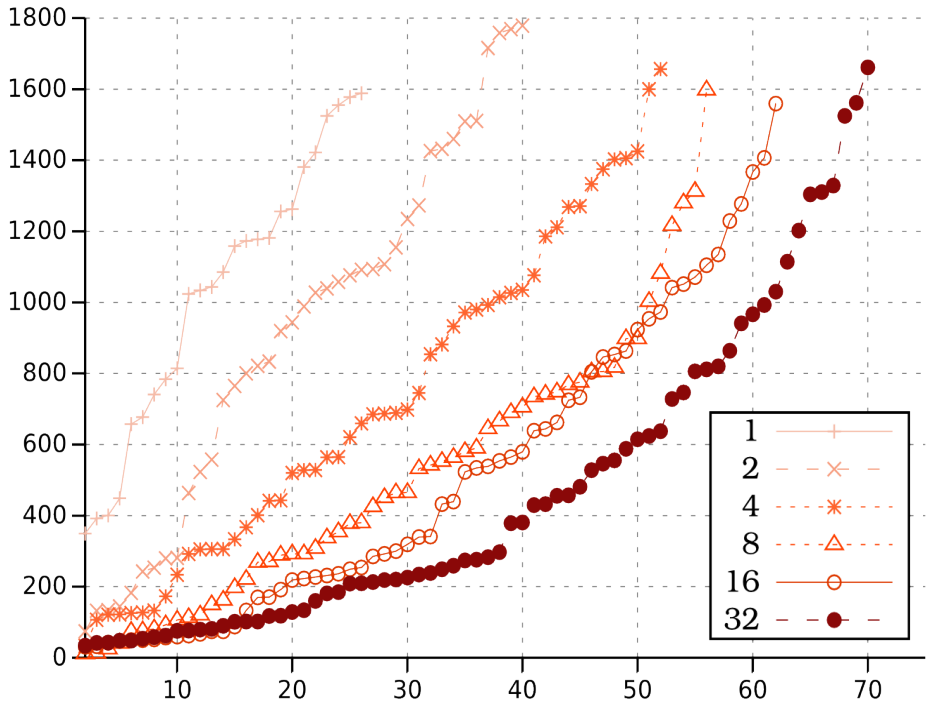


- x-axis : Number of imported clauses (sum over 8 threads)
- y-axis : Number of promoted clauses (sum over 8 threads)

Results

- 100 benchmarks of SAT'15 competition (parallel track)
- WC timeout is set to 1,800 seconds
- computer with 32 cores (quad-processor Intel XEON X7550)

#cores	SAT	UNS	Total
1	15	11	26
2	25	15	40
4	29	23	52
8	31	25	56
16	36	27	63
32	42	28	70



A word on D-Syrup

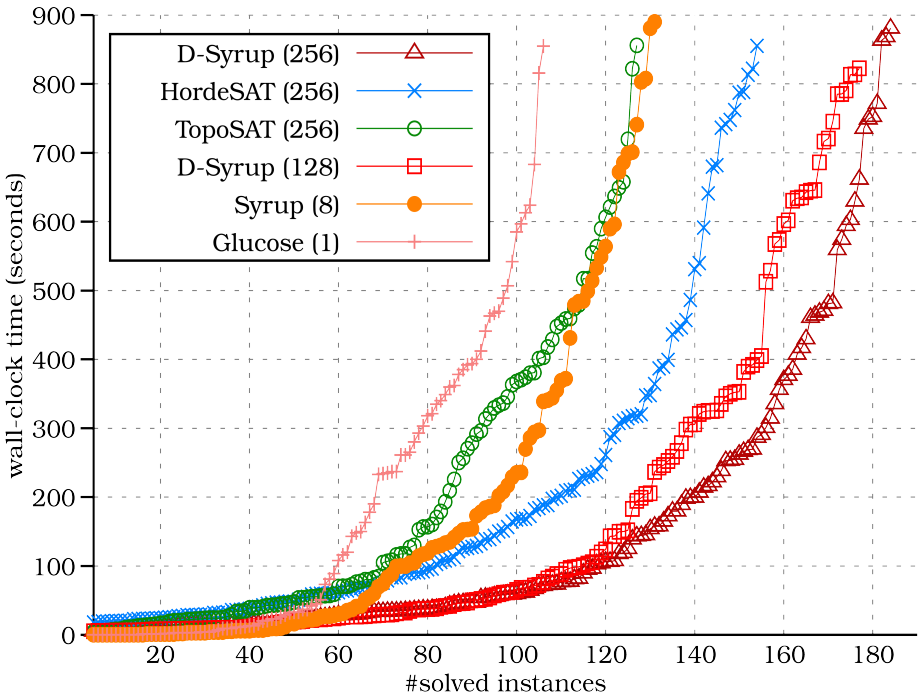
- syrup is scalable up to 32 cores
- How to add cores :
  - Using expensive computers ! (but always with a limited number of cores)
  - Deploy syrup on distributed architectures

- A distributed version of syrup
- Each computer embeds a Syrup solver
- Computers communicate in a decentralized way
- Export clauses using the same strategy. No dedicated heuristic
- Possible thanks to the library MPIch and a particular communication stratgegy

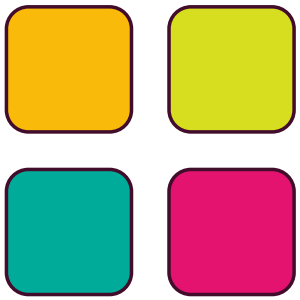
Results

- SAT'16 benchmarks
- 8 cores syrup (16 or 32 computers)

Solver	#cores	SAT	UNS	Total
D-syrup	256	75	109	184
HordeSat	256	70	84	154
TopoSat	256	69	58	127
D-Syrup	128	73	104	177
syrup	8	56	75	131
glucose	1	49	57	106



- syrup (8 cores) vs TopoSat (256 cores)
- TopoSat has good SAT results and really bad UNSAT ones ! (it does not share enough clauses and is penalized by the pure message passing model)



Incremental SAT solving

Incremental SAT solving : an Introduction

- A surprising effect of solvers' efficiency : used as NP-Complete oracles
  - IC3 : thousands of calls on *simple* formulas [Bradley 2012]
  - MUS extraction [Belov etal. 2012]
  - MaxSAT
- Many calls on similar instances
- CDCL solvers learn form the PAST ! !

Keep the solver alive

Minimum Unsatisfiable Subformula

$x \vee y \vee z$

$\neg x \vee y \vee z$

$\neg x \vee \neg y$

$x \vee \neg y$

$x \vee w$

$\neg x \vee \neg z$

UNSAT

$x \vee \neg z$

$w \vee z \vee \neg y$

$w \vee \neg x \vee \neg z$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses

Minimum Unsatisfiable Subformula

$x \vee y \vee z$	$x \vee \neg y$	$x \vee \neg z$
$\neg x \vee y \vee z$	$x \vee w$	$w \vee z \vee \neg y$
$\neg x \vee \neg y$	$\neg x \vee \neg z$	$w \vee \neg x \vee \neg z$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses

Minimum Unsatisfiable Subformula

$x \vee y \vee z$ $x \vee \neg y$ $x \vee \neg z$

$\neg x \vee y \vee z$ $x \vee w$ $w \vee z \vee \neg y$

$\neg x \vee \neg y$ $\neg x \vee \neg z$ $w \vee \neg x \vee \neg z$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula

$$\neg x \vee y \vee z$$
$$\neg x \vee \neg y$$
$$x \vee \neg y$$
$$x \vee w$$
$$\neg x \vee \neg z$$

SAT

$$x \vee \neg z$$
$$w \vee z \vee \neg y$$
$$w \vee \neg x \vee \neg z$$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula

$x \vee y \vee z$	$x \vee \neg y$	$x \vee \neg z$
$\neg x \vee y \vee z$	$x \vee w$	$w \vee z \vee \neg y$
$\neg x \vee \neg y$	$\neg x \vee \neg z$	$w \vee \neg x \vee \neg z$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula

$x \vee y \vee z$	$x \vee \neg y$	$x \vee \neg z$
$\neg x \vee y \vee z$	$x \vee w$	$w \vee z \vee \neg y$
$\neg x \vee \neg y$	$\neg x \vee \neg z$	$w \vee \neg x \vee \neg z$

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula

$x \vee y \vee z$	$x \vee \neg y$	$x \vee \neg z$
$\neg x \vee y \vee z$		$w \vee z \vee \neg y$
$\neg x \vee \neg y$	$\neg x \vee \neg z$	$w \vee \neg x \vee \neg z$
	UNSAT	

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula

$x \vee y \vee z$

$\neg x \vee y \vee z$

$x \vee \neg y$

$\neg x \vee \neg y$

$x \vee \neg z$

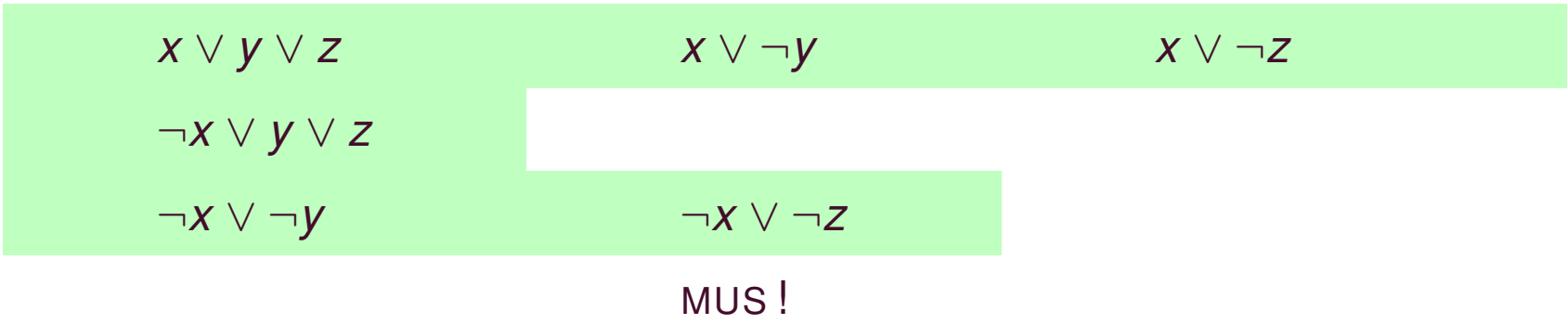
$\neg x \vee \neg z$

$w \vee \neg x \vee \neg z$

UNSAT

- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Minimum Unsatisfiable Subformula



- The formula is inconsistant : Why ?
- Minimal unsatisfiable subset of clauses
- Different approaches
  - Local search [Piette et al, ECAI 2006]
  - Resolution based [Nadel, FMCAD 2010]
  - Constructive or destructive [Belov etal, AI Com 2012]. The tool MUSER

Working with Assumptions

- A formula F
- A set of assumptions, $\ell_1, \ell_2, \ldots, \ell_n$ with ℓ_i are (*fresh*) literals
- Solve $F \wedge \ell_1 \wedge \ell_2 \ldots \wedge \ell_n$
- Incremental SAT solving : the process can be repeated with new assumptions

Working with Assumptions

- A formula F
- A set of assumptions, $\ell_1, \ell_2, \ldots, \ell_n$ with ℓ_i are (*fresh*) literals
- Solve $F \wedge \ell_1 \wedge \ell_2 \ldots \wedge \ell_n$
- Incremental SAT solving : the process can be repeated with new assumptions

First solution

- Simplify : $F' = F \wedge \ell_1 \wedge \ell_2 \ldots \wedge \ell_n$
- Solve F'
- Learnt clauses can not be kept

Working with Assumptions

- A formula F
- A set of assumptions, $\ell_1, \ell_2, \ldots, \ell_n$ with ℓ_i are (*fresh*) literals
- Solve $F \wedge \ell_1 \wedge \ell_2 \ldots \wedge \ell_n$
- Incremental SAT solving : the process can be repeated with new assumptions

Working with Assumptions

- A formula F
- A set of assumptions, $\ell_1, \ell_2, \ldots, \ell_n$ with ℓ_i are (*fresh*) literals
- Solve $F \wedge \ell_1 \wedge \ell_2 \ldots \wedge \ell_n$
- Incremental SAT solving : the process can be repeated with new assumptions

Second Solution

- First, selects all assumptions as decision variables :
  one level => one assumption
- Second, Run the SAT solver as usual
- All learnt clauses can be kept
- One can explain unsatisfiability wrt set of assumptions !

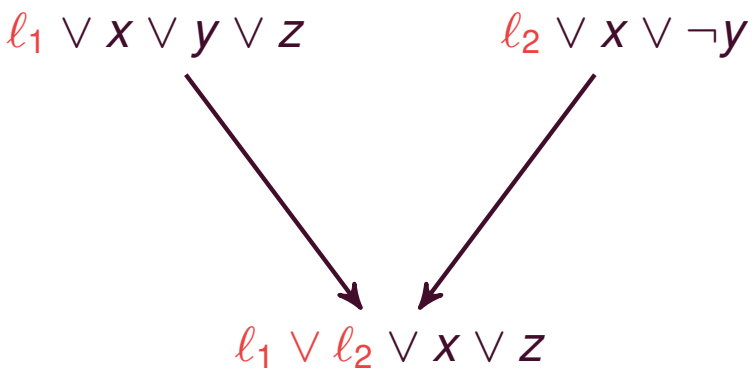
Forget some clauses

- Add one selector (fresh variable) ℓ_i per clause

$\ell_1 \vee x \vee y \vee z$
$\ell_4 \vee \neg x \vee y \vee z$
$\ell_7 \vee \neg x \vee \neg y$

$\ell_2 \vee x \vee \neg y$
$\ell_5 \vee x \vee w$
$\ell_8 \vee \neg x \vee \neg z$

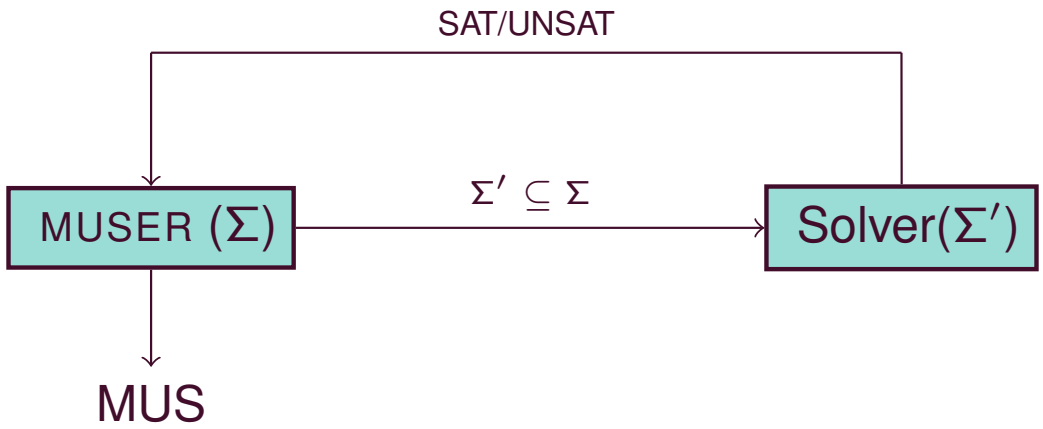
$\ell_3 \vee x \vee \neg z$
$\ell_6 \vee w \vee z \vee \neg y$
$\ell_9 \vee w \vee \neg x \vee \neg z$



- Learnt clause contains selectors of all original clauses used to generate it

Muser Architecture

Incremental SAT



- One of the best MUS extractor
- Successive calls to a SAT oracle
- Non independent calls
- Informations between two calls are preserved
  - Heuristics : VSIDS, phase saving, restarts...
  - **Learnt clauses**

Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$\ell_1 \vee x \vee y \vee z$
$\ell_2 \vee x \vee \neg y$
$\ell_3 \vee x \vee \neg z$
$\ell_4 \vee \neg x \vee y \vee z$
$\ell_5 \vee x \vee w$
$\ell_6 \vee w \vee z \vee \neg y$
$\ell_7 \vee \neg x \vee \neg y$
$\ell_8 \vee \neg x \vee \neg z$
$\ell_9 \vee w \vee \neg x \vee \neg z$

$\ell_1 \vee \ell_2 \vee x \vee z$

Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$\ell_1 \vee x \vee y \vee z$
$\ell_2 \vee x \vee \neg y$
$\ell_3 \vee x \vee \neg z$
$\ell_4 \vee \neg x \vee y \vee z$
$\ell_5 \vee x \vee w$
$\ell_6 \vee w \vee z \vee \neg y$
$\ell_7 \vee \neg x \vee \neg y$
$\ell_8 \vee \neg x \vee \neg z$
$\ell_9 \vee w \vee \neg x \vee \neg z$

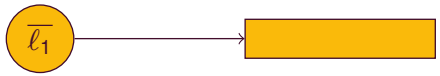
$\ell_1 \vee \ell_2 \vee x \vee z$

Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$$
\begin{aligned}
& x \vee y \vee z \\
\ell_2 \vee & x \vee \neg y \\
\ell_3 \vee & x \vee \neg z \\
\ell_4 \vee & \neg x \vee y \vee z \\
\ell_5 \vee & x \vee w \\
\ell_6 \vee & w \vee z \vee \neg y \\
\ell_7 \vee & \neg x \vee \neg y \\
\ell_8 \vee & \neg x \vee \neg z \\
\ell_9 \vee & w \vee \neg x \vee \neg z \\
\\
\ell_1 \vee \ell_2 \vee & x \vee z
\end{aligned}
$$

DL 1



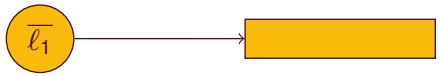
Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$$
\begin{aligned}
& x \vee y \vee z \\
\ell_2 \vee & x \vee \neg y \\
\ell_3 \vee & x \vee \neg z \\
\ell_4 \vee & \neg x \vee y \vee z \\
\ell_5 \vee & x \vee w \\
\ell_6 \vee & w \vee z \vee \neg y \\
\ell_7 \vee & \neg x \vee \neg y \\
\ell_8 \vee & \neg x \vee \neg z \\
\ell_9 \vee & w \vee \neg x \vee \neg z
\end{aligned}
$$

$$\ell_2 \vee x \vee z$$

DL 1

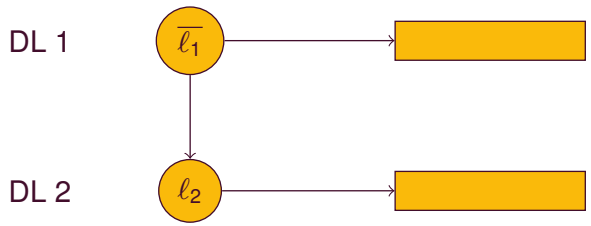


Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$$x \vee y \vee z$$
$$\ell_2 \vee x \vee \neg y$$
$$\ell_3 \vee x \vee \neg z$$
$$\ell_4 \vee \neg x \vee y \vee z$$
$$\ell_5 \vee x \vee w$$
$$\ell_6 \vee w \vee z \vee \neg y$$
$$\ell_7 \vee \neg x \vee \neg y$$
$$\ell_8 \vee \neg x \vee \neg z$$
$$\ell_9 \vee w \vee \neg x \vee \neg z$$

$$\ell_2 \vee x \vee z$$



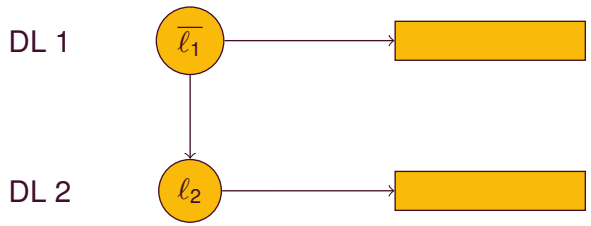
Forget Some Clauses

- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$$x \vee y \vee z$$

$$\ell_3 \vee x \vee \neg z$$
$$\ell_4 \vee \neg x \vee y \vee z$$
$$\ell_5 \vee x \vee w$$
$$\ell_6 \vee w \vee z \vee \neg y$$
$$\ell_7 \vee \neg x \vee \neg y$$
$$\ell_8 \vee \neg x \vee \neg z$$
$$\ell_9 \vee w \vee \neg x \vee \neg z$$

$$\ell_2 \vee x \vee z$$

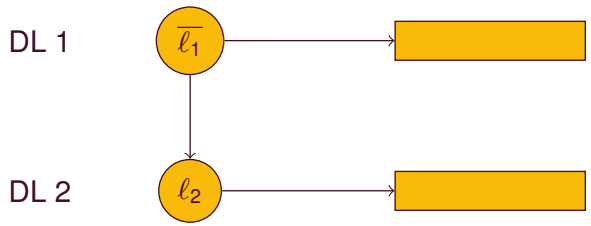


Forget Some Clauses

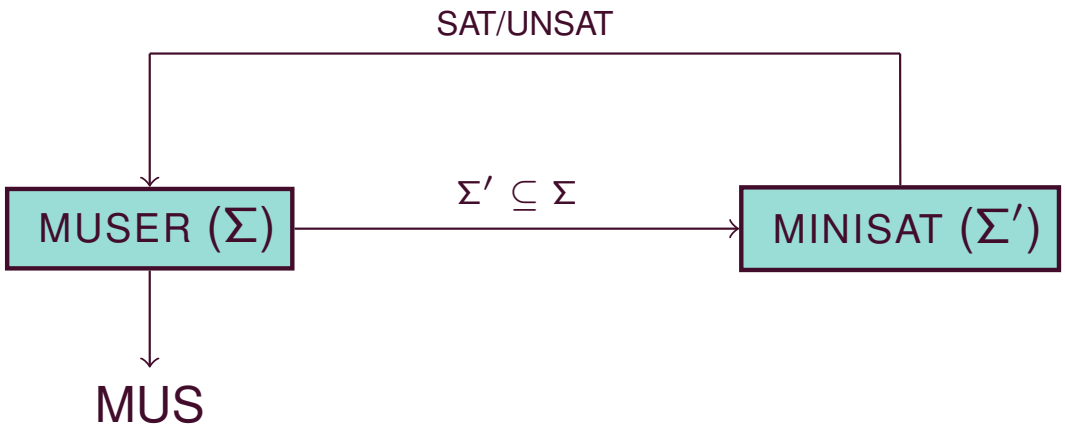
- Assign ℓ_i (as an assumption) to false to activate the clause i
- Assign ℓ_j (as an assumption) to true to disable the clause j
- All learnt clauses related to the clause j a disable clause are disabled too !

$$x \vee y \vee z$$

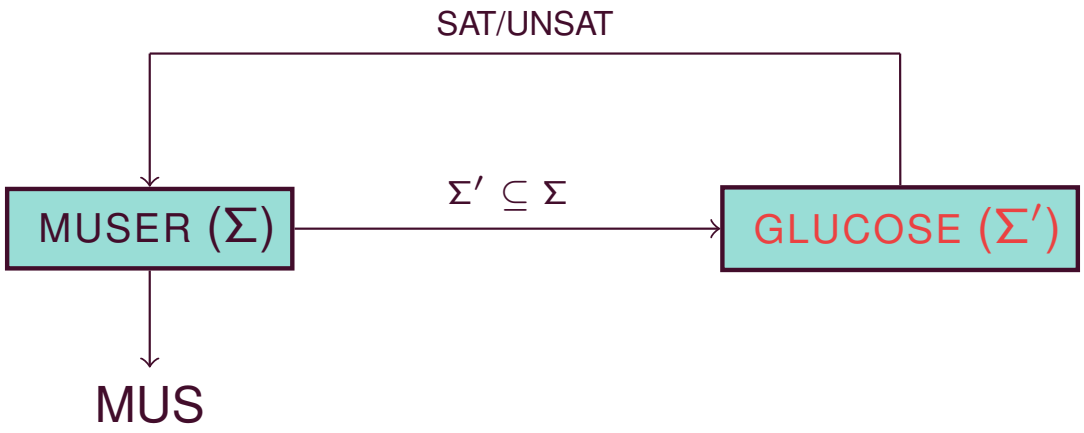
$$\ell_3 \vee x \vee \neg z$$
$$\ell_4 \vee \neg x \vee y \vee z$$
$$\ell_5 \vee x \vee w$$
$$\ell_6 \vee w \vee z \vee \neg y$$
$$\ell_7 \vee \neg x \vee \neg y$$
$$\ell_8 \vee \neg x \vee \neg z$$
$$\ell_9 \vee w \vee \neg x \vee \neg z$$



Glucose inside Muser



Glucose inside Muser



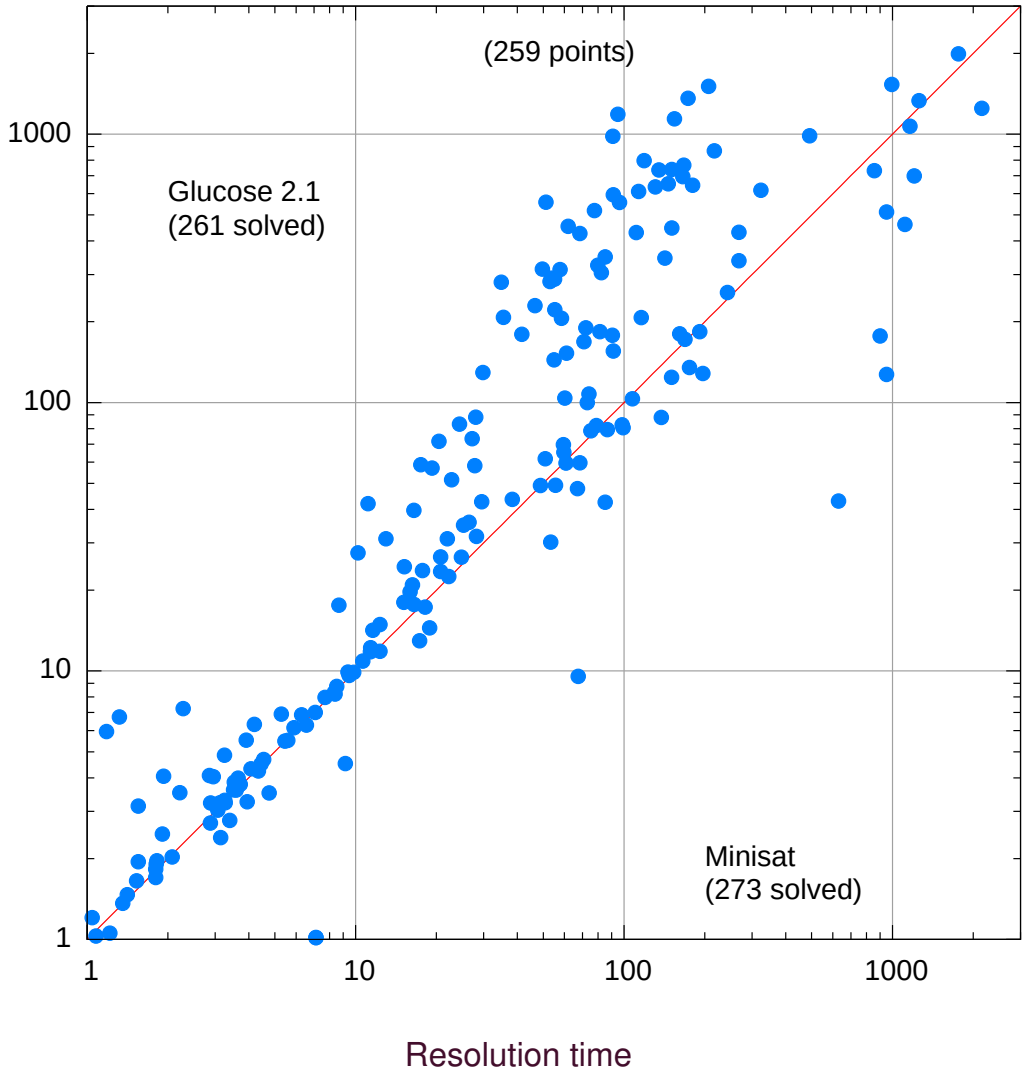
- Plug GLUCOSE in MUSER
- Adapt and modify GLUCOSE to improve MUSER performances

Improve SAT oracle in order to improve the MUSER tool

Test Set

- 300 instances from the SAT competition 2011, MUS category
- timeout set to 2400 seconds
- MUSER is used with default options (destructive approach, model rotation)

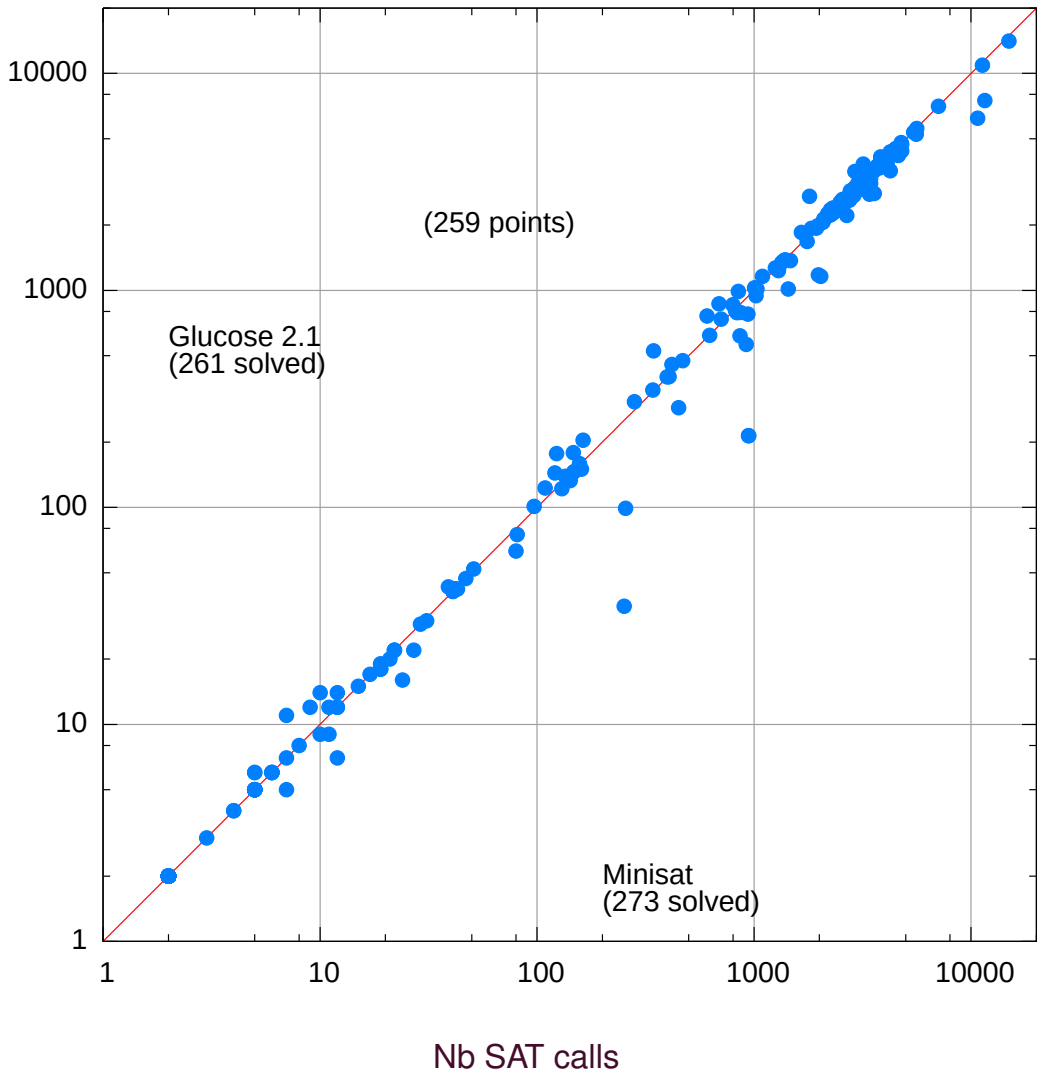
A First Attempt



Disappointing Results

Trying to explain these bad results

Disappointing Results



Disappointing Results

Trying to explain these bad results

- Comparable number of oracle calls
- Easy SAT calls
- Difficult UNSAT ones
- GLUCOSE is supposed to be good on UNSAT formulas

Disappointing Results

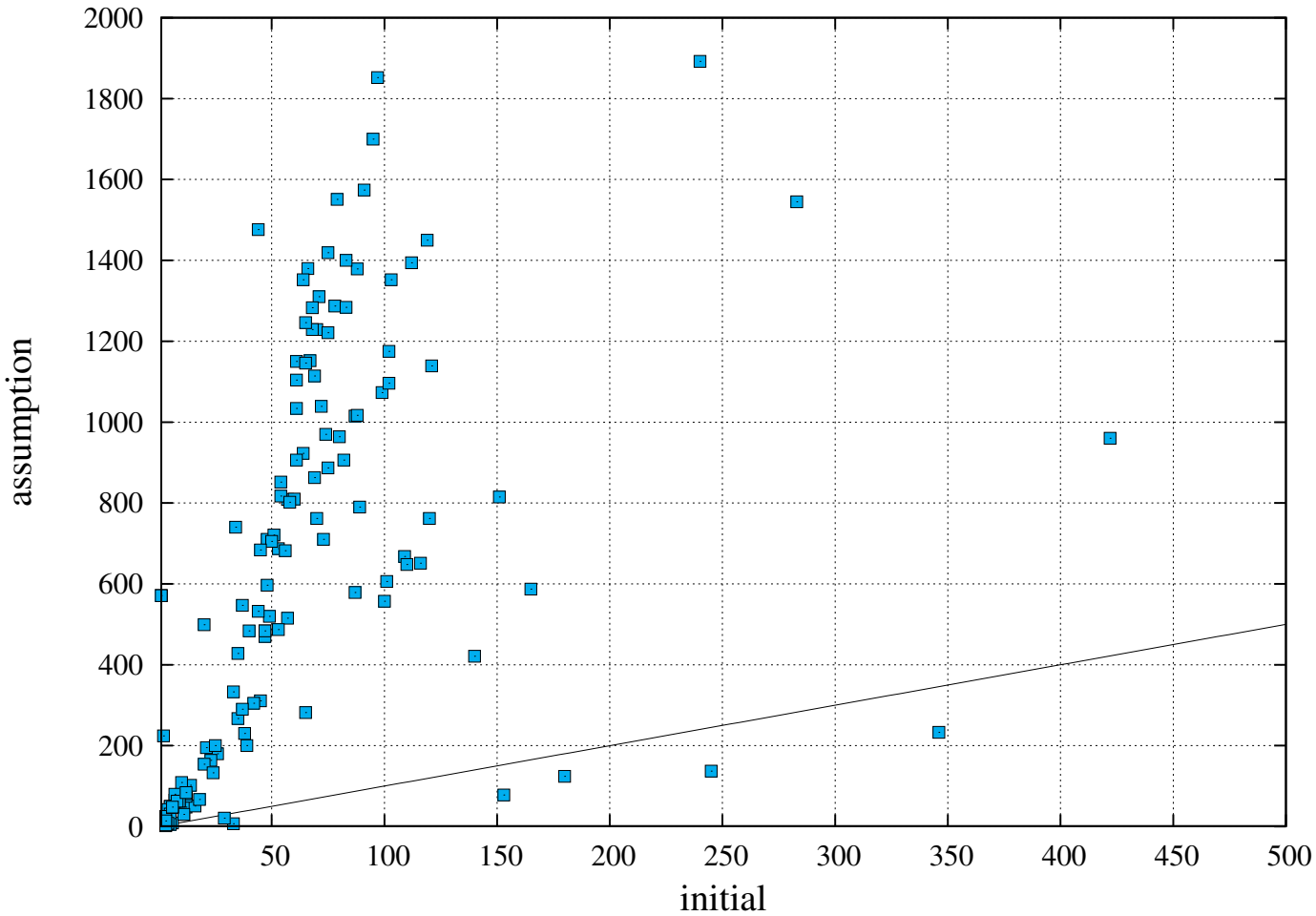
Trying to explain these bad results

- Comparable number of oracle calls
- Easy SAT calls
- Difficult UNSAT ones
- GLUCOSE is supposed to be good on UNSAT formulas

- GLUCOSE uses LBD for cleaning, restarts...
- Each assumption uses its own decision level

Disappointing Results

- Each point represents an instance
- x-axis is the average number of initial variables in learnt clauses
- y-axis is the average number of selector variables in learnt clauses



Disappointing Results

		LBD				
			size		LBD	
Instance	#C	time	avg	max	avg	max
fdmus_b21_96	8541	29	1145	5980	1095	5945
longmult6	8853	46	694	3104	672	3013
dump_vc950	360419	110	522	36309	498	35873
g7n	70492	190	1098	16338	1049	16268

- LBD looks like size
- Clauses are very long

Disappointing Results

Trying to explain these bad results

- Comparable number of oracle calls
- Easy SAT calls
- Difficult UNSAT ones
- GLUCOSE is supposed to be good on UNSAT formulas

- GLUCOSE uses LBD for cleaning, restarts...
- Each assumption uses its own decision level
- **The LBD of a clause looks like its size !**

Disappointing Results

Trying to explain these bad results

- Comparable number of oracle calls
- Easy SAT calls
- Difficult UNSAT ones
- GLUCOSE is supposed to be good on UNSAT formulas

- GLUCOSE uses LBD for cleaning, restarts...
- Each assumption uses its own decision level
- **The LBD of a clause looks like its size !**

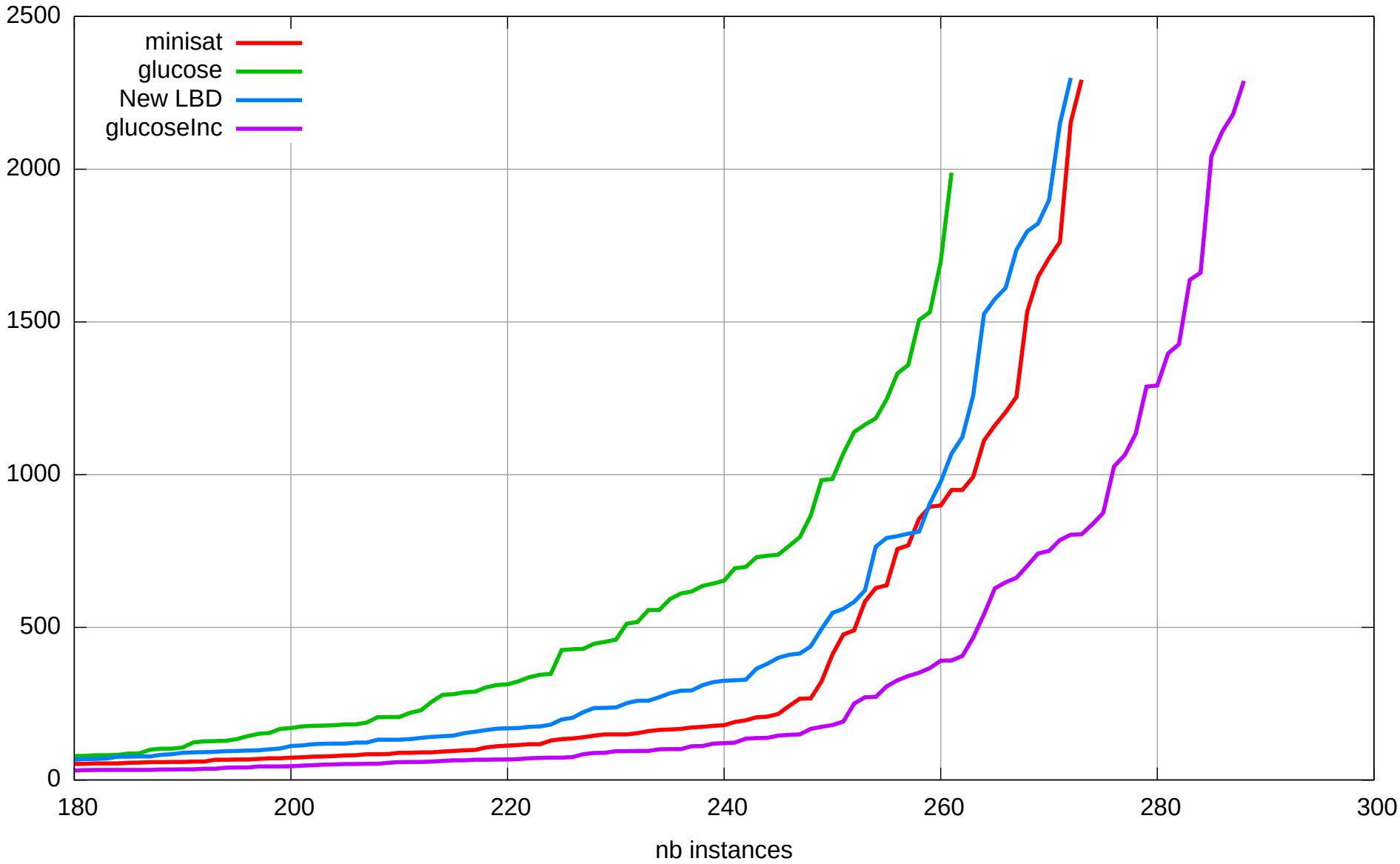
Refine LBD : Do not take into account selectors

Clauses are too Long

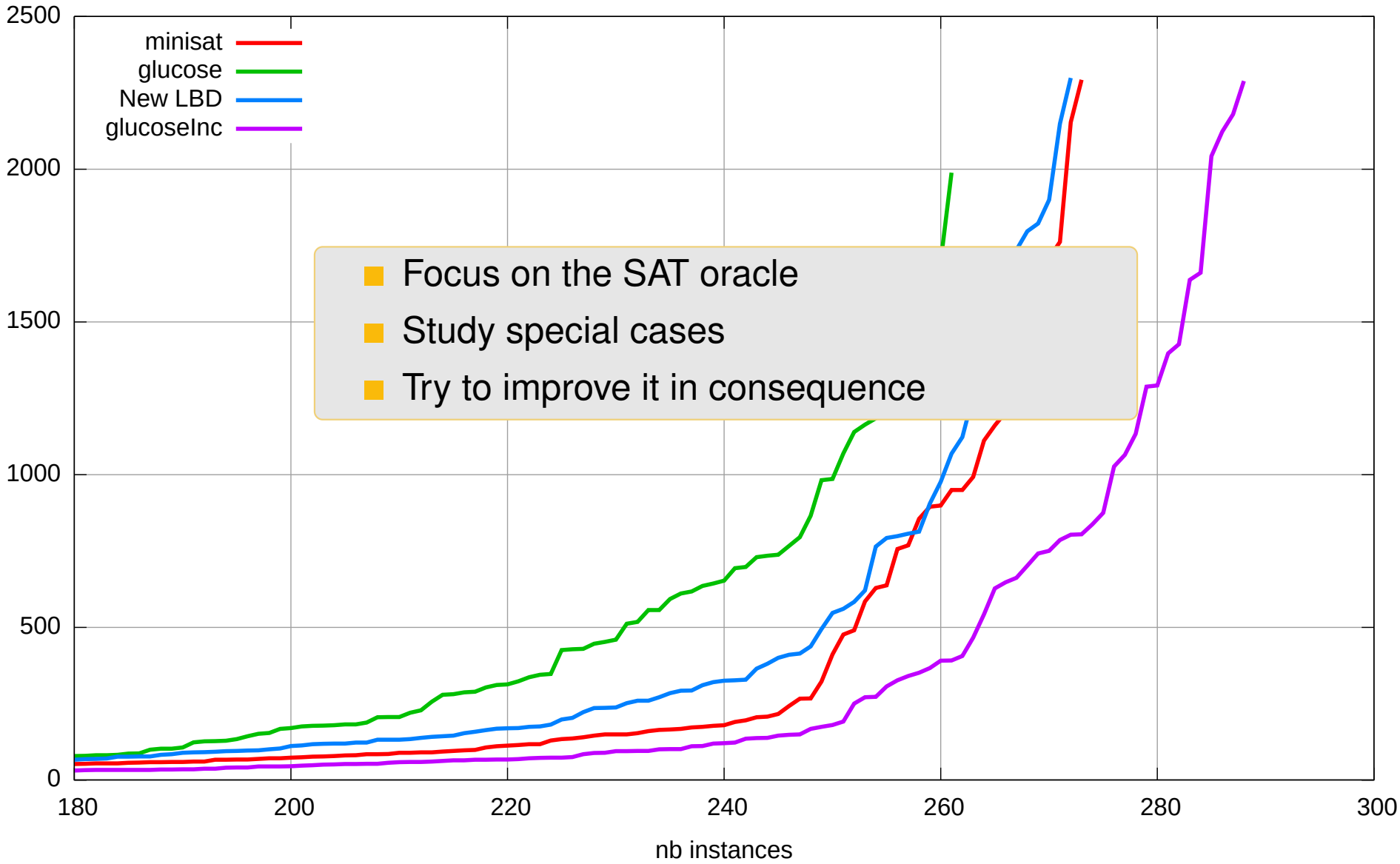
Many algorithms need to traverse clauses

- Dynamic computing of LBD (useful but costly)
  - → Store the number of selectors in the clause
  - → Stop when all initial literals have been tested
- Conflict analysis
  - → Force initial literals to be placed at the beginning
- Unit propagation
  - → Look for a non selector literal or a satisfied one
  - → Push selectors at the end of the clause
- Deleting satisfiable clauses
  - → Take only watched literals into account

Comparison

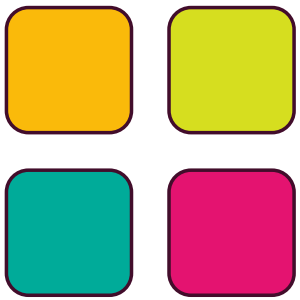

minisat
glucose
New LBD
glucoseInc
2500
2000
1500
1000
500
0
180
200
220
240
260
280
300
nb instances

Comparison



- Focus on the SAT oracle
- Study special cases
- Try to improve it in consequence

- Take a look at a CDCL solver (the essentials of Minisat in fact) : assumptions branch



Conclusion

Conclusion

SAT CAN HELP YOU !