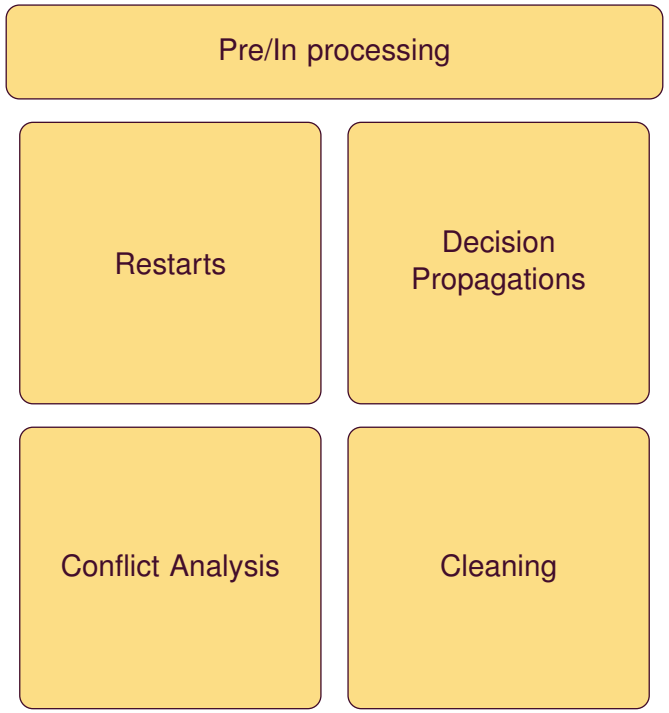
SAT solver essentials, SAT modeling
CDCL

Gilles Audemard

VTSA School - Liege - 2021

Thanks to N. Szczepanski and L. Simon

CDCL Architecture



Sequence of decision, propagations

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

Sequence of decision, propagations

DL 1

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

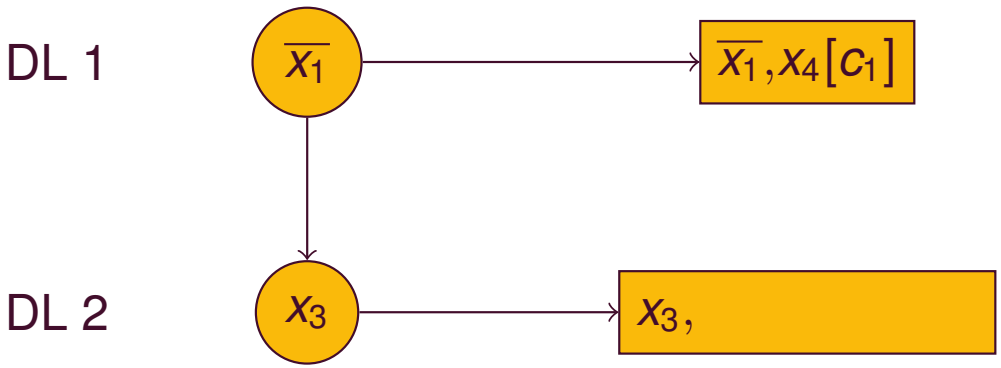
Sequence of decision, propagations

DL 1

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

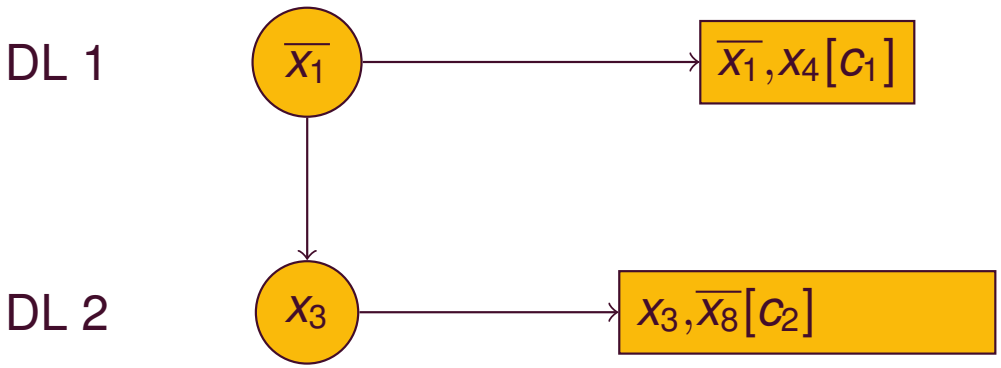
Sequence of decision, propagations

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



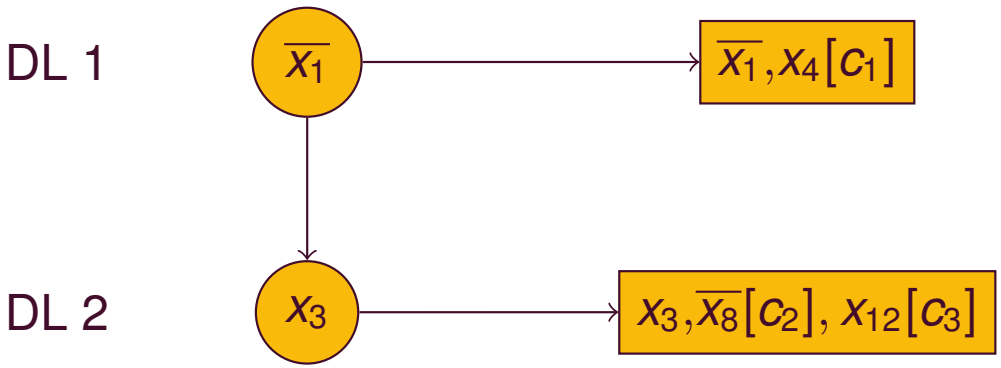
Sequence of decision, propagations

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



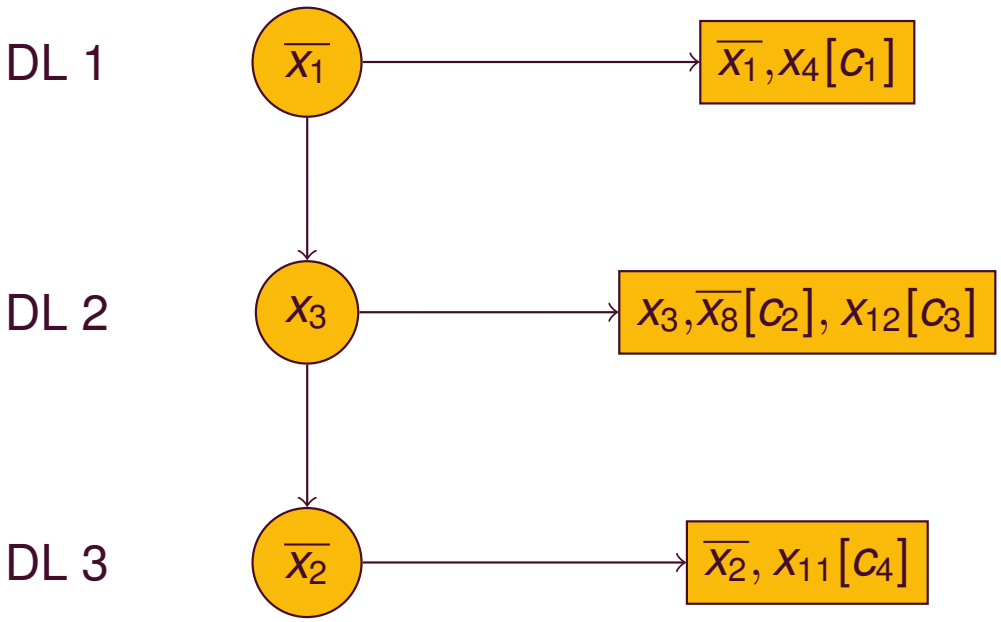
Sequence of decision, propagations

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



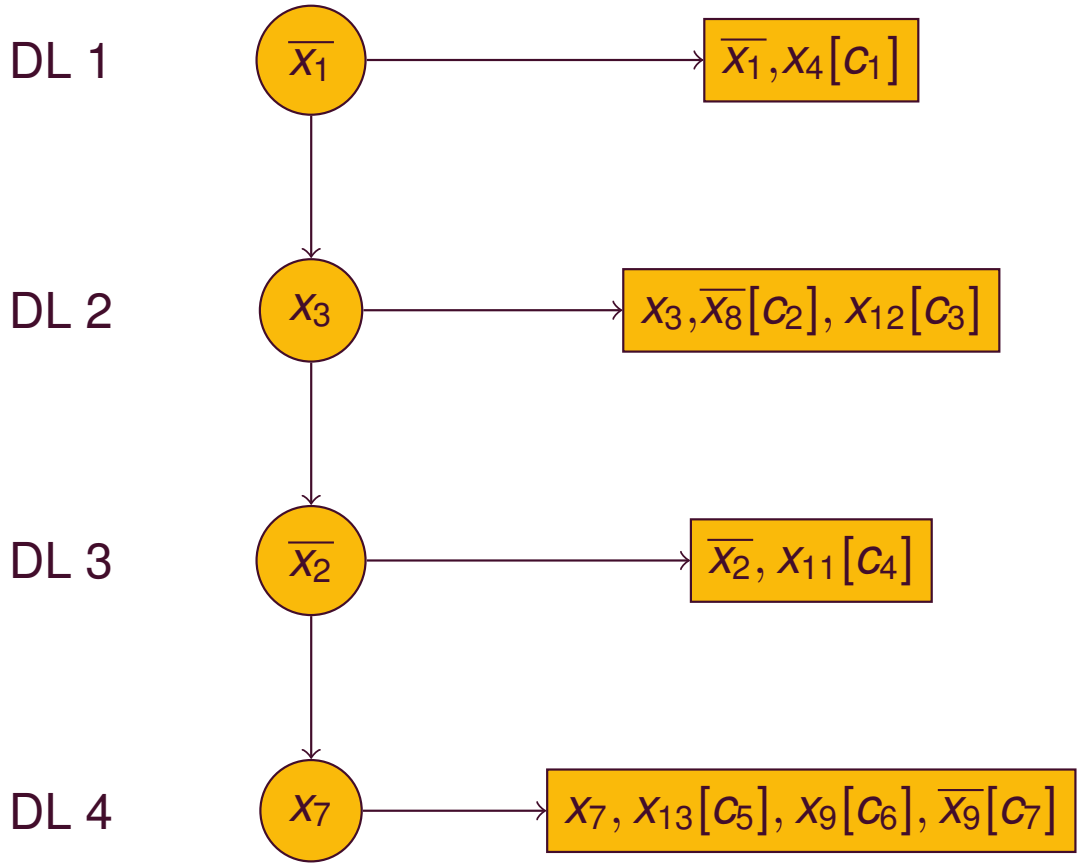
Sequence of decision, propagations

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



Sequence of decision, propagations

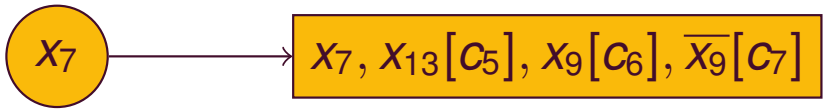
$c_1 = x_1 \vee x_4$
$c_2 = x_1 \vee \overline{x_3} \vee \overline{x_8}$
$c_3 = x_1 \vee x_8 \vee x_{12}$
$c_4 = x_2 \vee x_{11}$
$c_5 = \overline{x_3} \vee \overline{x_7} \vee x_{13}$
$c_6 = \overline{x_3} \vee \overline{x_7} \vee \overline{x_{13}} \vee x_9$
$c_7 = x_8 \vee \overline{x_7} \vee \overline{x_9}$



Conflict analysis

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

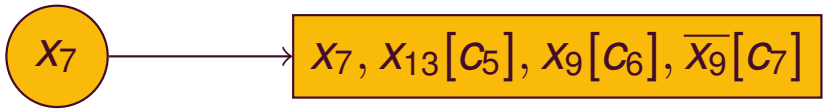
DL 4



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

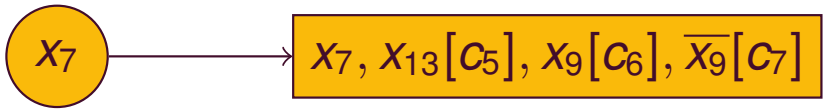


$$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$$

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

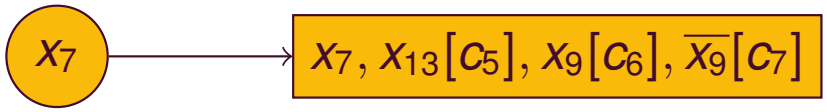


$$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$$
$$d_1 = d^* \otimes_{x_{13}} c_5 = \overline{x_3} \vee x_8 \vee \overline{x_7}$$

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



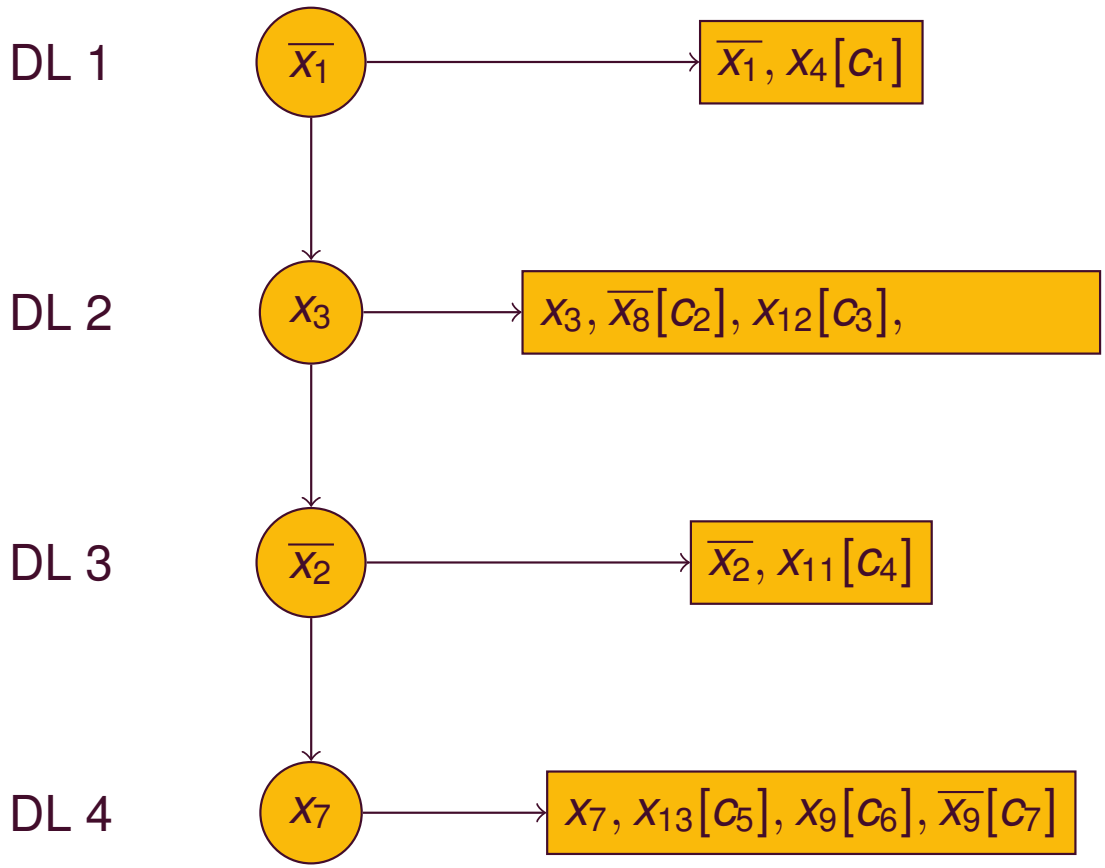
$$d^* = c_7 \otimes_{x_9} c_6 = \overline{x_3} \vee x_8 \vee \overline{x_7} \vee \overline{x_{13}}$$
$$d_1 = d^* \otimes_{x_{13}} c_5 = \overline{x_3} \vee x_8 \vee \overline{x_7}$$

- 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...

Backjumping

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

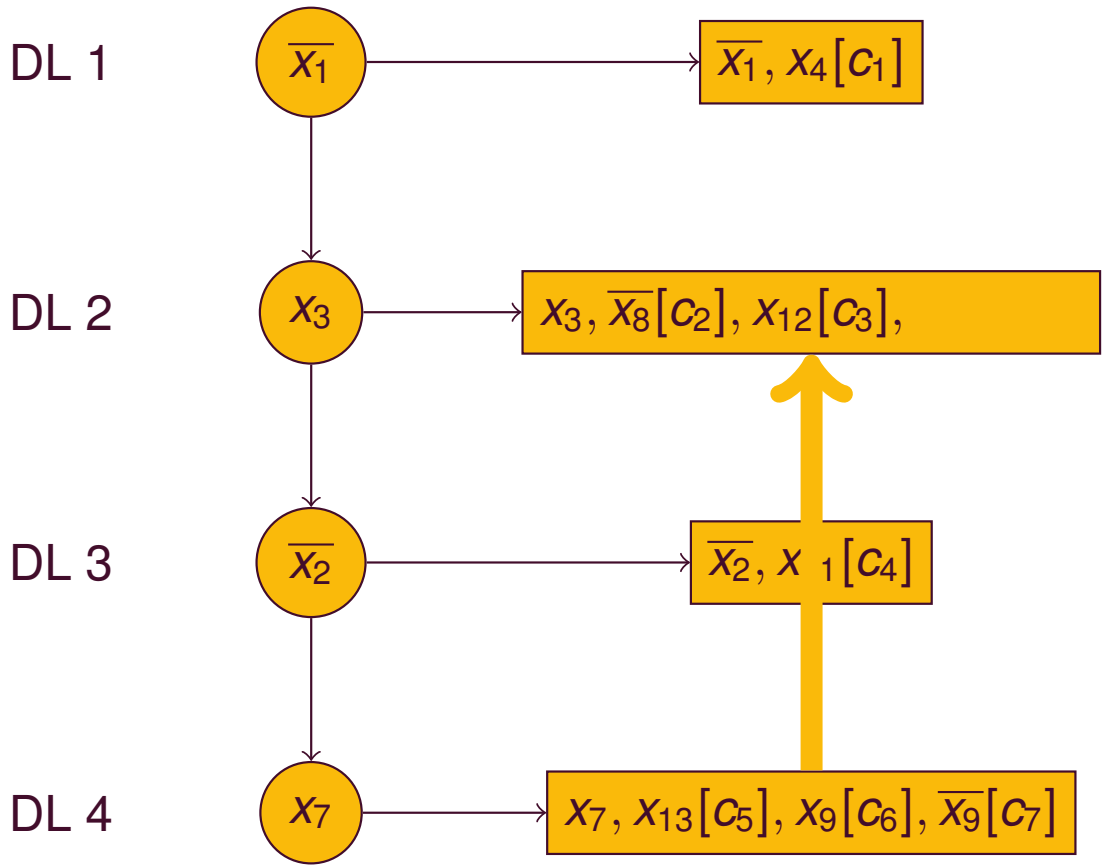
$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$



Backjumping

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

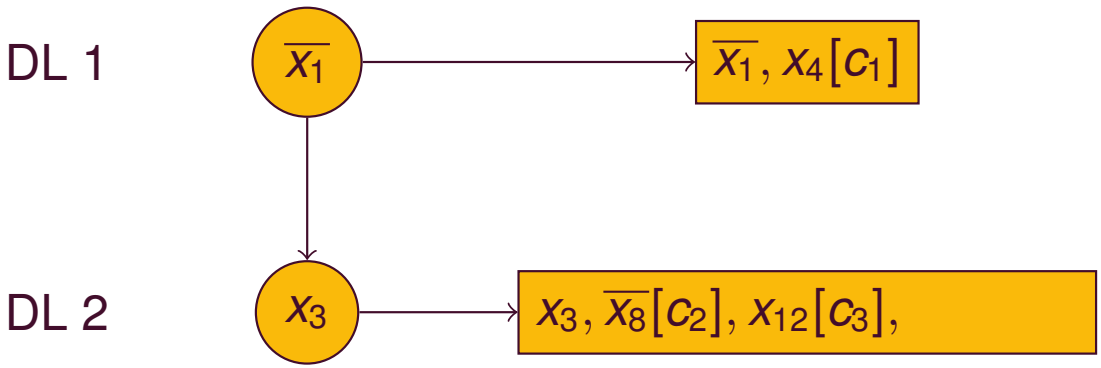
$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$



Backjumping

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

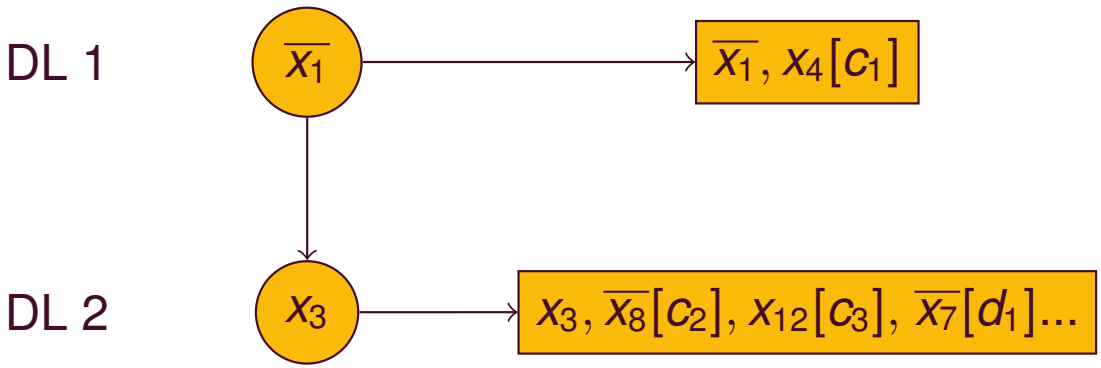
$d_1 = \overline{x_3} \vee x_8 \vee \overline{x_7}$



Backjumping

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

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



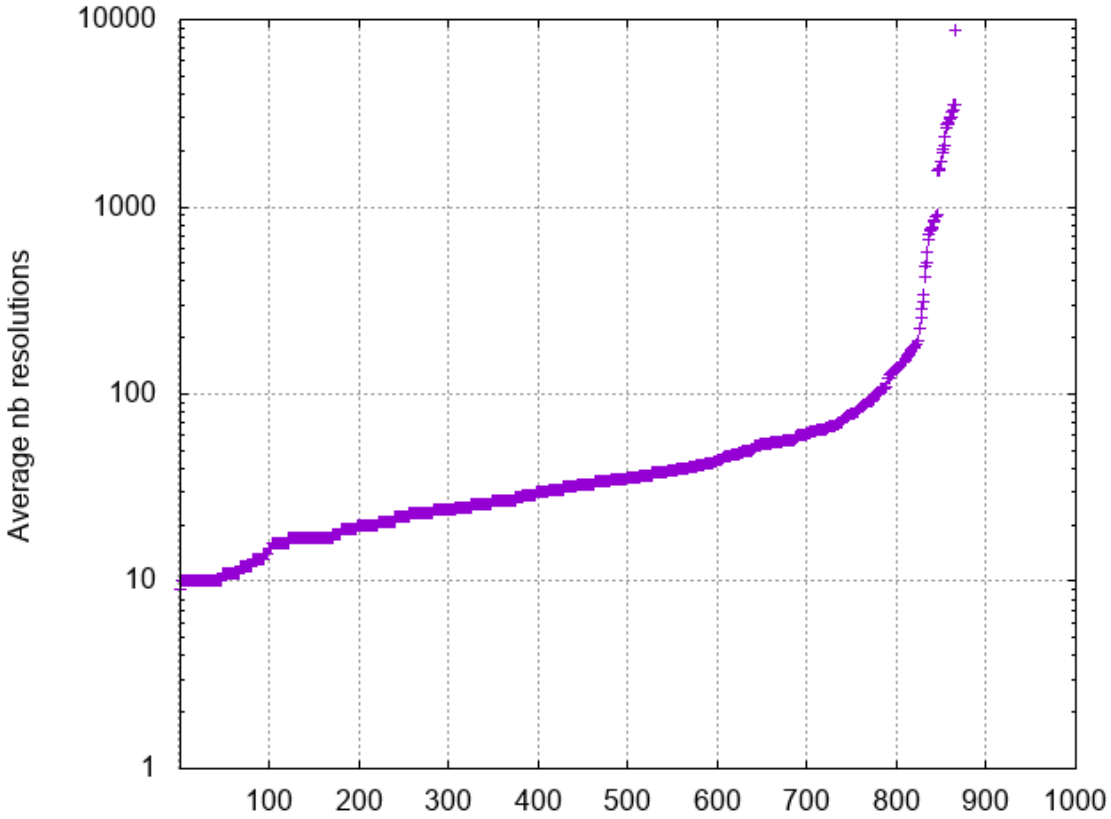
Exercise

- A CNF formula

$c_1 = a \vee d$
$c_2 = b \vee h$
$c_3 = \neg c \vee \neg e \vee \neg i \vee g$
$c_4 = a \vee \neg c \vee \neg f$
$c_5 = \neg c \vee \neg e \vee i$
$c_6 = \neg d \vee j \vee f$
$c_7 = \neg i \vee \neg j \vee \neg g$

1. Using the decision literal (in this order) $\neg a, c, \neg b, e$, mimic the set of unit propagations
2. Once a conflict is reached, extract the assertive clause (1-UIP) using some resolution steps
3. What are the consequences if we learn this clause ?

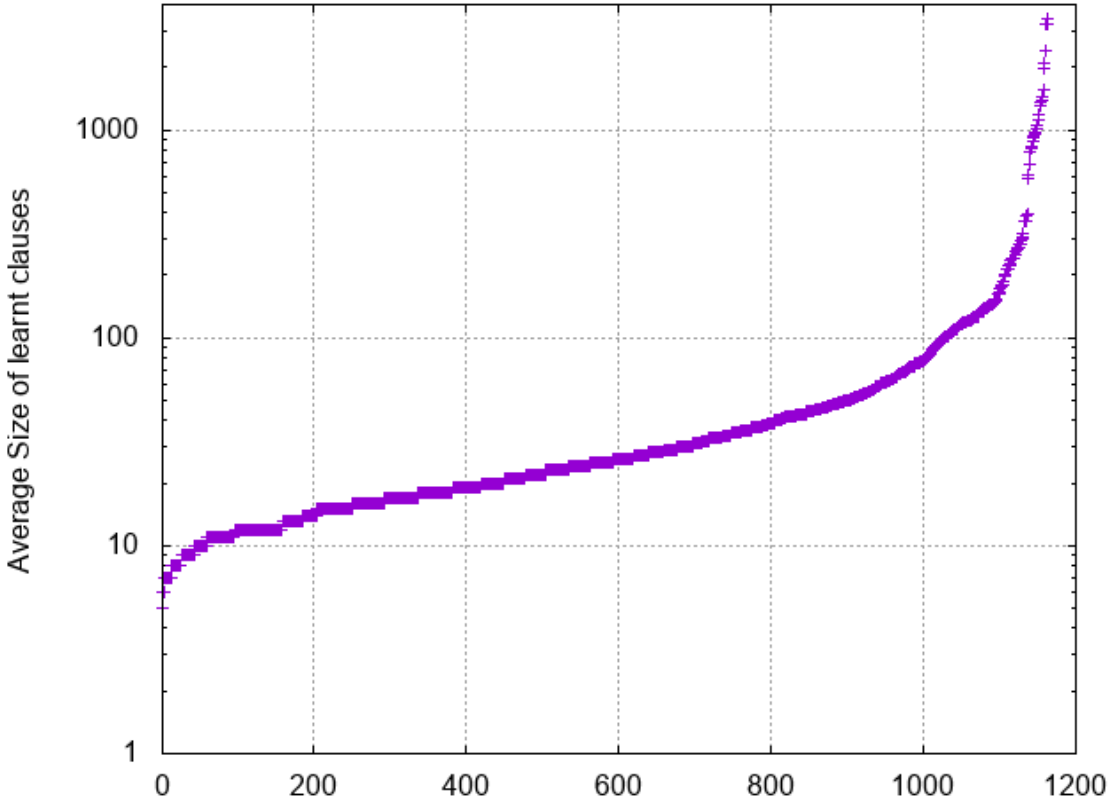
Many resolutions



stats after 10,000 conflicts on 900 benchmarks

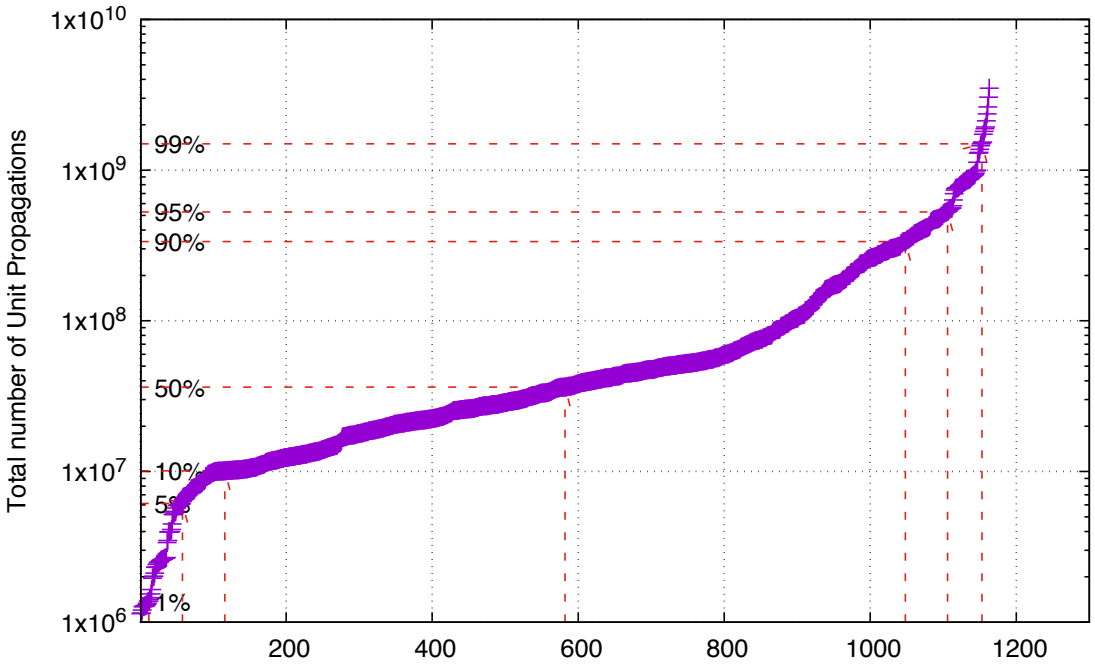
- Many resolutions on correlated clauses
- Only one clause stored !
- One of the reason of the efficiency of learning

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

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

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

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

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

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

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

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

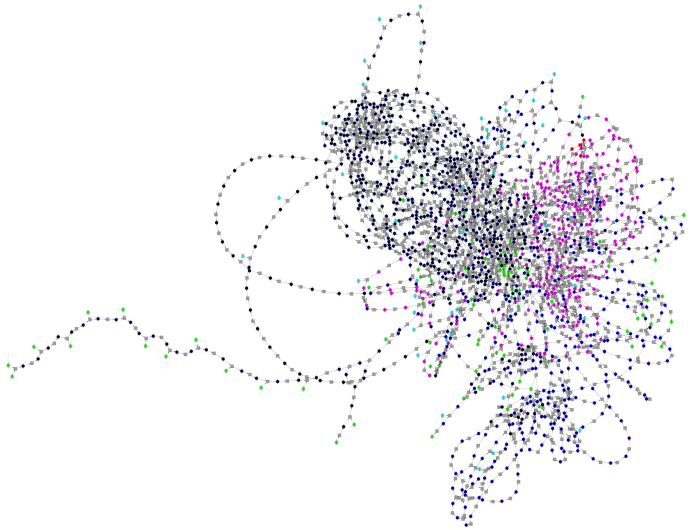
- Store the current size of the clause and the current state
- When propagating x_1,
  - take a look to all clauses where $\neg x_1$ appears (c_2), Update size in consequence
  - take a look to all clauses where x_1 appears (c_1, c_3), set the status SAT to these clauses
- When backtracking, restore counters and status of clauses

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_2	ℓ_3	ℓ_4	ℓ_5

- The first two literals are the witnesses



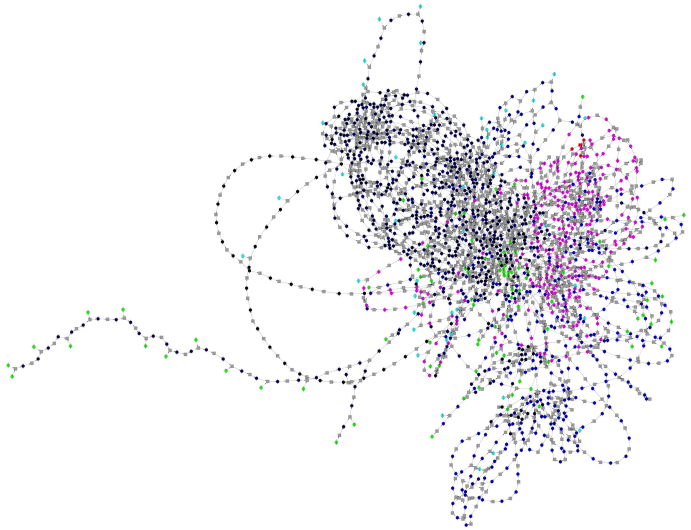
Real Implication Graph

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)

ℓ_1	ℓ_2	ℓ_3	ℓ_4	ℓ_5

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



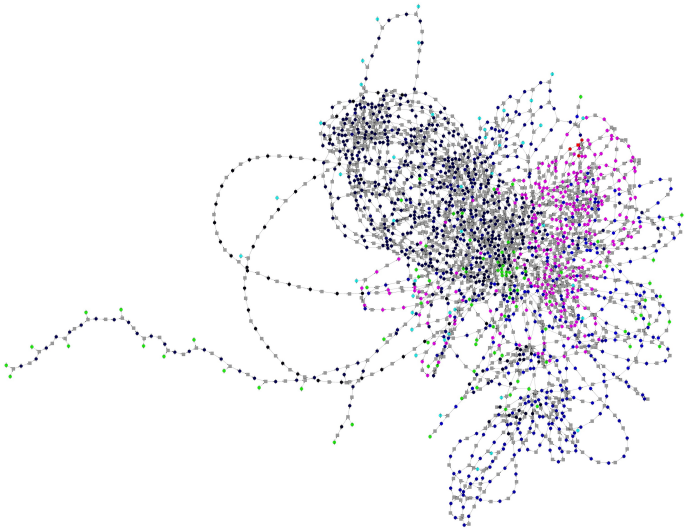
Real Implication Graph

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)

ℓ_1	ℓ_2	ℓ_3	ℓ_4	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



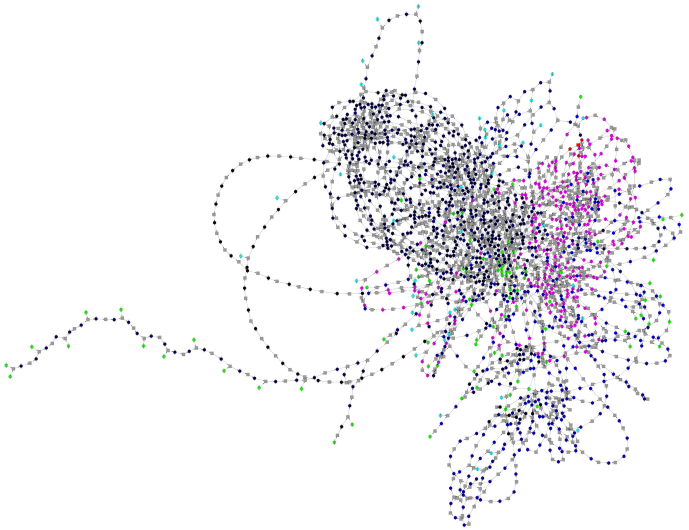
Real Implication Graph

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_2	ℓ_3	ℓ_4	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



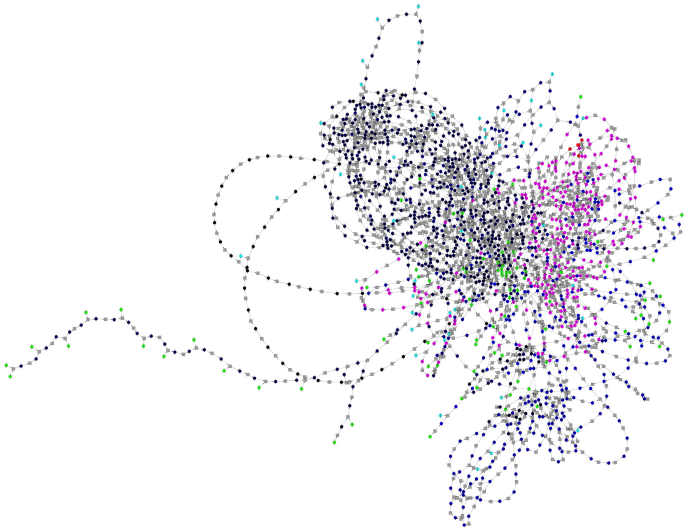
Real Implication Graph

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



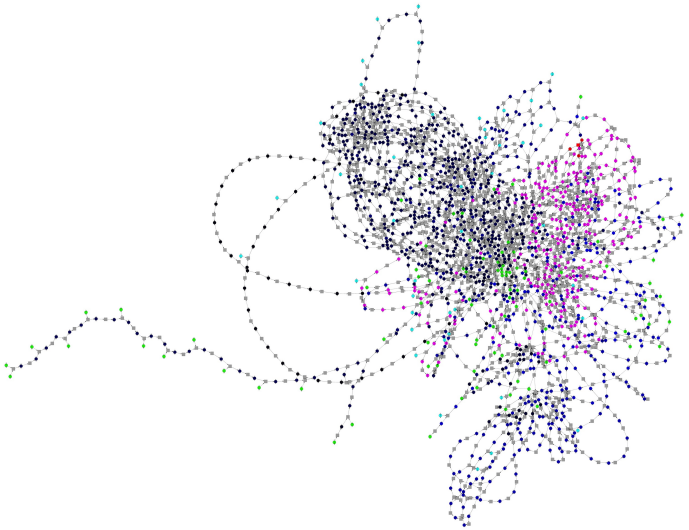
Real Implication Graph

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



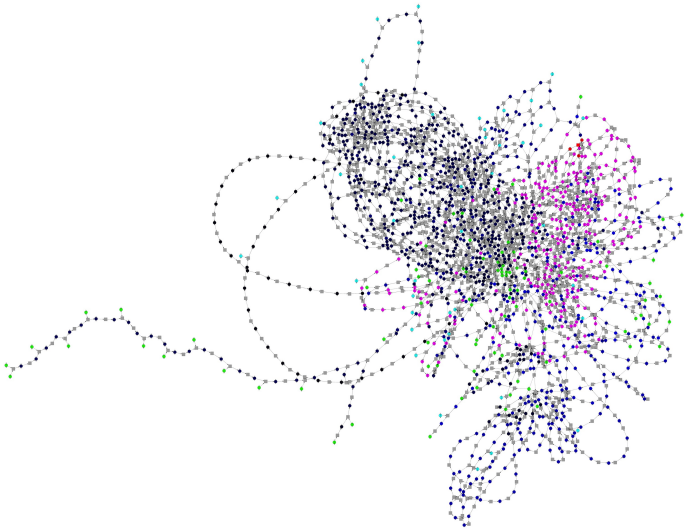
Real Implication Graph

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.



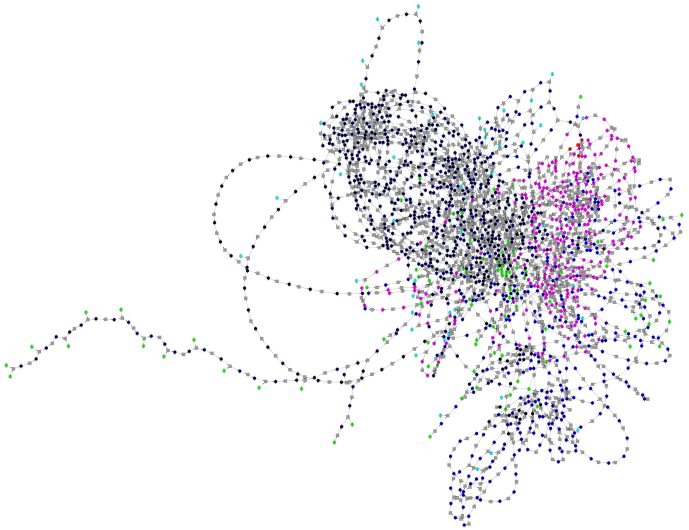
Real Implication Graph

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 !!!!!



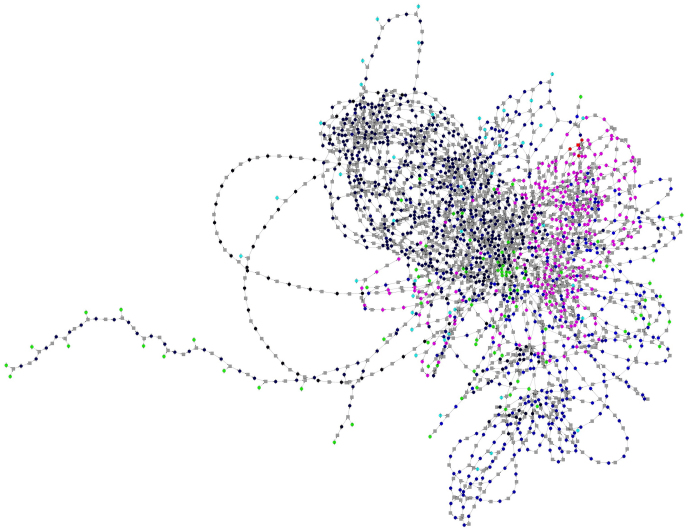
Real Implication Graph

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.



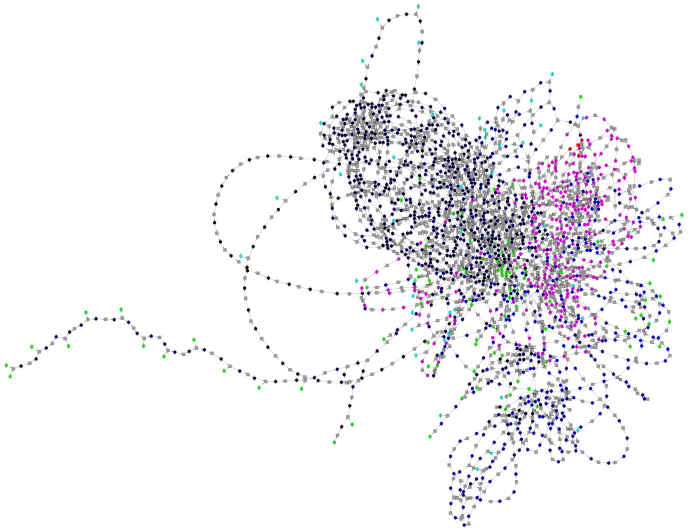
Real Implication Graph

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



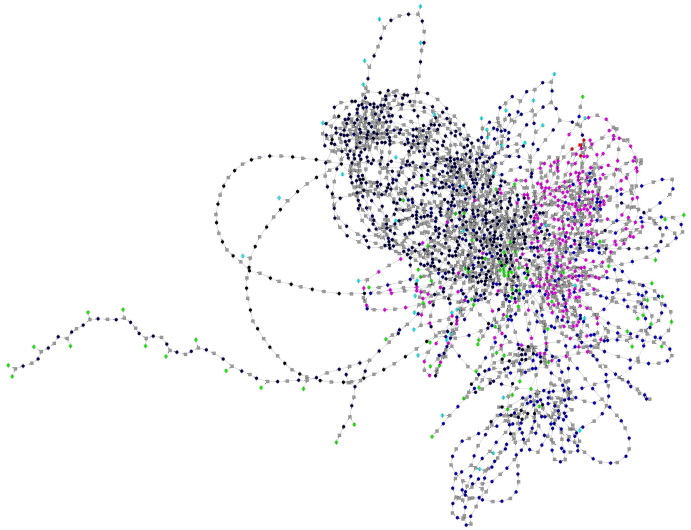
Real Implication Graph

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

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

$$x_1 \rightarrow \{c_1, c_3\} \qquad x_4 \rightarrow \{c_1\}$$
$$\neg x_1 \rightarrow \{c_2\} \qquad \neg x_4 \rightarrow \{\}$$

$$x_2 \rightarrow \{c_2, c_4\} \qquad x_5 \rightarrow \{c_4\}$$
$$\neg x_2 \rightarrow \{c_3\} \qquad \neg x_5 \rightarrow \{\}$$

$$x_3 \rightarrow \{\} \qquad x_6 \rightarrow \{\}$$
$$\neg x_3 \rightarrow \{\} \qquad \neg x_6 \rightarrow \{\}$$

- When x_1 is propagated, take a look to all clauses watched by $\neg x_1$ (here just 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 appear **recently** in conflict/analysis

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

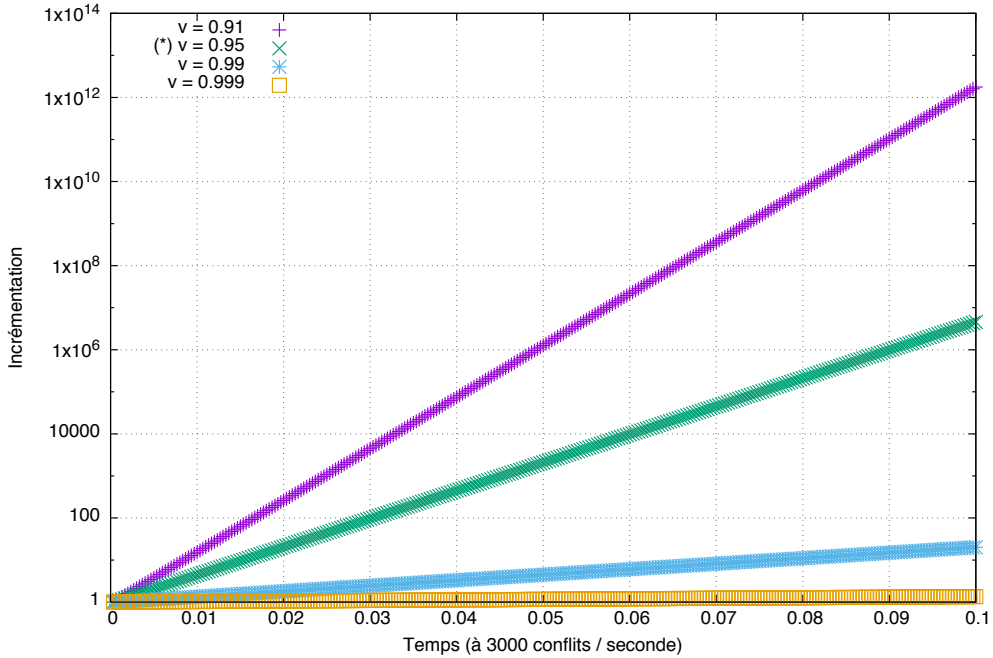
VSIDS : Variable State Independent Decaying Sum [Moskewicz2001]

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 then 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....
- After 100 conflicts $b = 131$, after 200 $b = 17292$, after 300 $b > 2000000$
- A good variable can become 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]

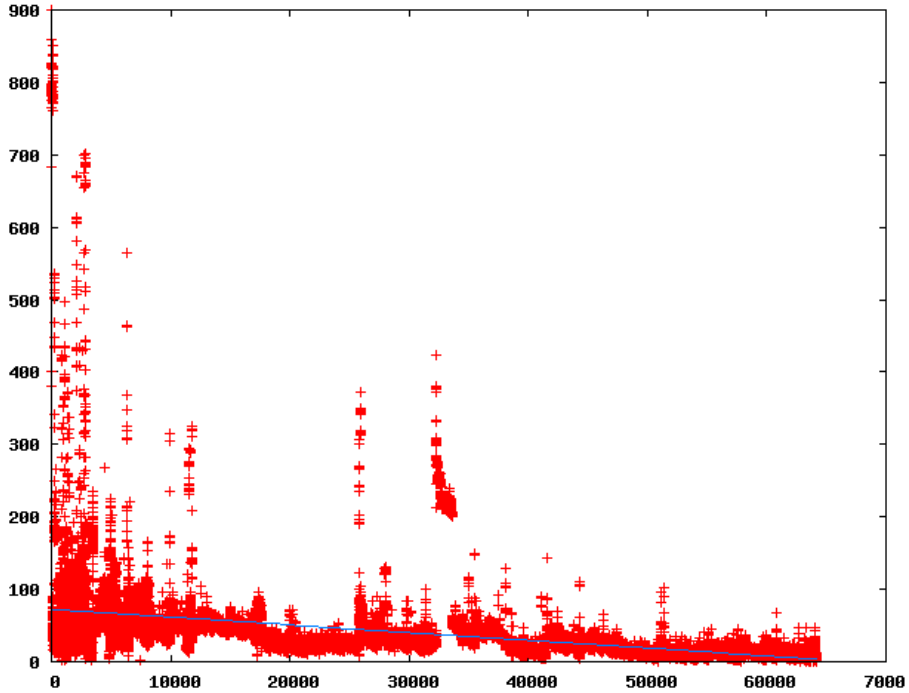
Intensive experiments is in the heart of GLUCOSE

- 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 from SAT'16 to SAT'20

Main Features

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

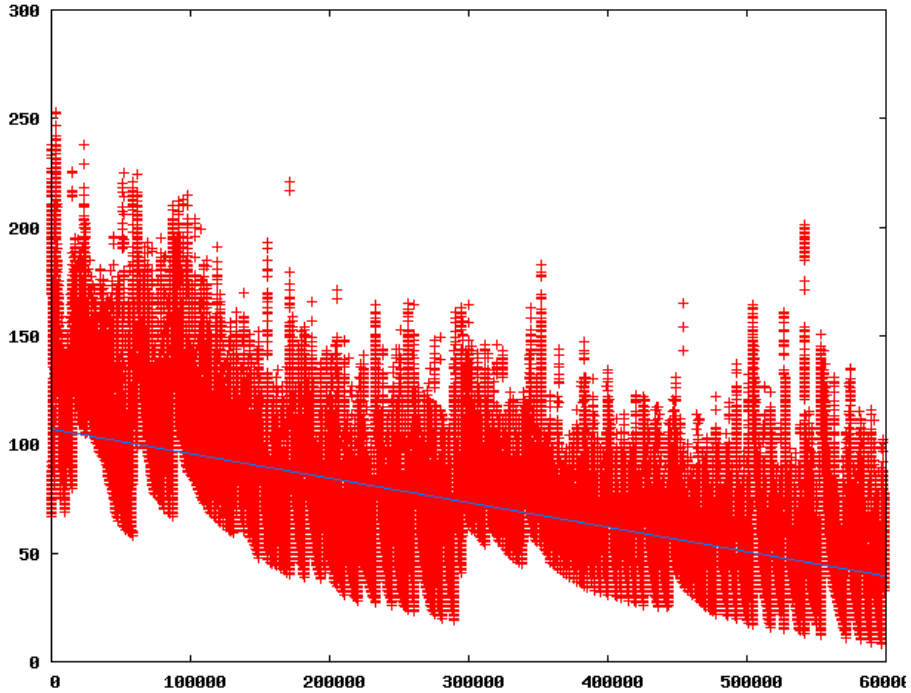
Many experiments behind Glucose . . .



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

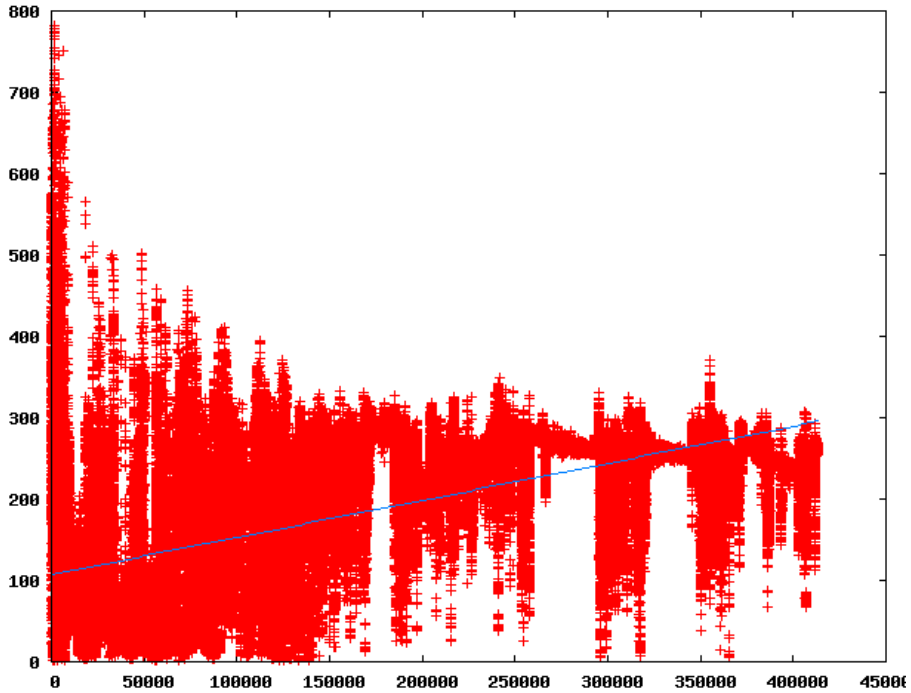
Many experiments behind Glucose . . .



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 + VSIDS

- 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 don't 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
  - Luby's 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, the threshols is increased

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

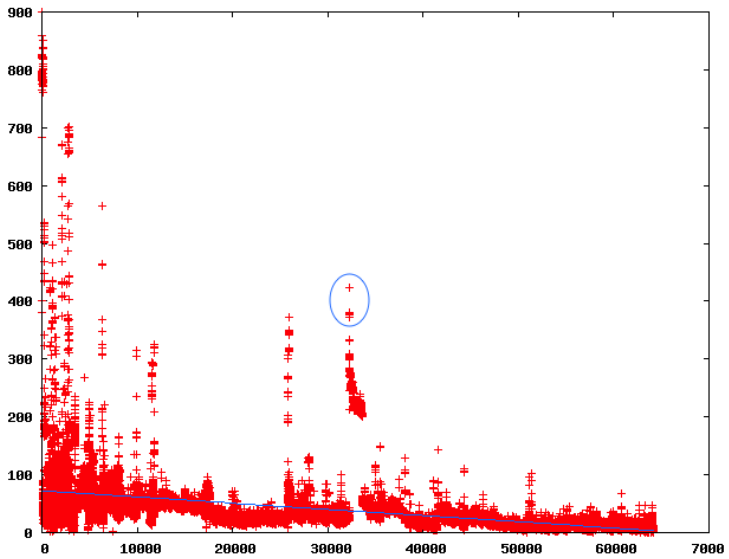
- Glucose aims to produce glue clauses
- If recent learnt clauses are bad (big LBD) a restart is performed
- We compare the global average of LBD and the current one (last X conflicts)
- We use
  - bounded queue (of size X) called `queueLBD`
  - the sum of all LBD clauses `sumLBD`

```
// 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

Some recent techniques

Vivification

- Initial idea : ”Vivifying Propositional Clausal Formulae”. Piette, Hamadi, Sais, ECAI 2008.
- Adapted to CDCL solvers in 2017 : ”An effective Learnt Clause Minimisation for CDCL SAT Solvers”. Luo etal, IJCAI.

- Take a clause $c = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$
- Assign $\neg\ell_1$, perform BCP
- if UNSAT, remove ℓ_1 from clause c
- iterate
- Simple, but efficient

Heuristic LRB

- Learning Rate Based Branching Heuristic for SAT Solvers. Liang, Ganesh, Poupart, Czarnecki, SAT 2016
- A novel heuristic, around 15 years after VSIDS
- Based on MAB
- Take into account the last time a variable was part of a conflict

Stable Focus Random walk

- ”Target Phases”. Biere and Fleury, 2020, POS
- Alternate between intensification and diversification
- Focus Mode : as usual
- Stable mode : do not forget the past (VSIDS constant=0.75). Alternate between different phase selection (True, False, Best, Random)
- Add a local search solver that search for a solution starting the best trail
- Use ticks (mem accesses (See Knuth)) as a measure to change the mode
- Very efficient

- Take a look to glucose Reboot

Learnt clause minimisation

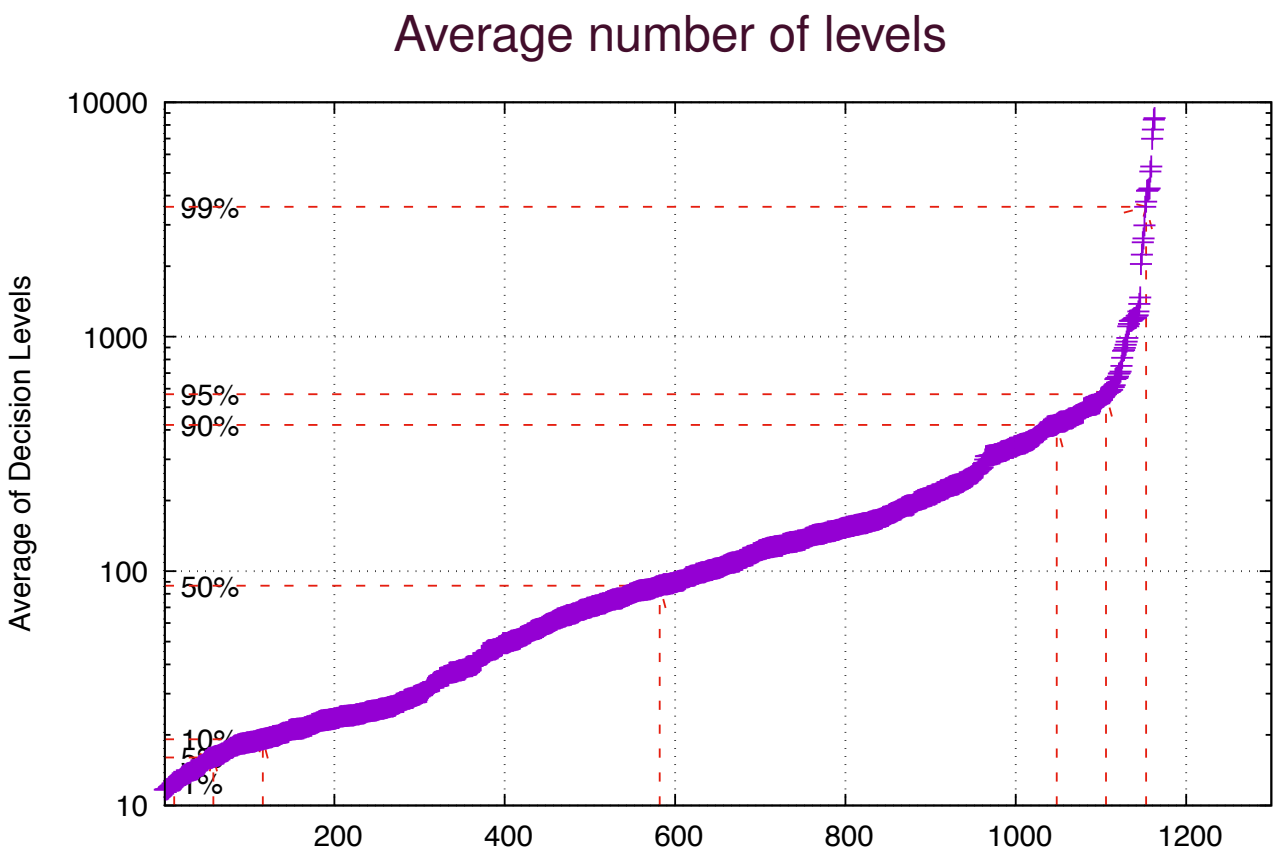
- Clause Size Reduction with all-UIP Learning. Feng, Bacchus, SAT20.
- Efficient All-UIP Learned Clause Minimization. Fleury, Biere, SAT21.
- Use ALL-UIP to reduce the size of the learnt clause
- Based on Self-sumbsumption

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

Behaviour of CDCL solvers

Outliers everywhere !

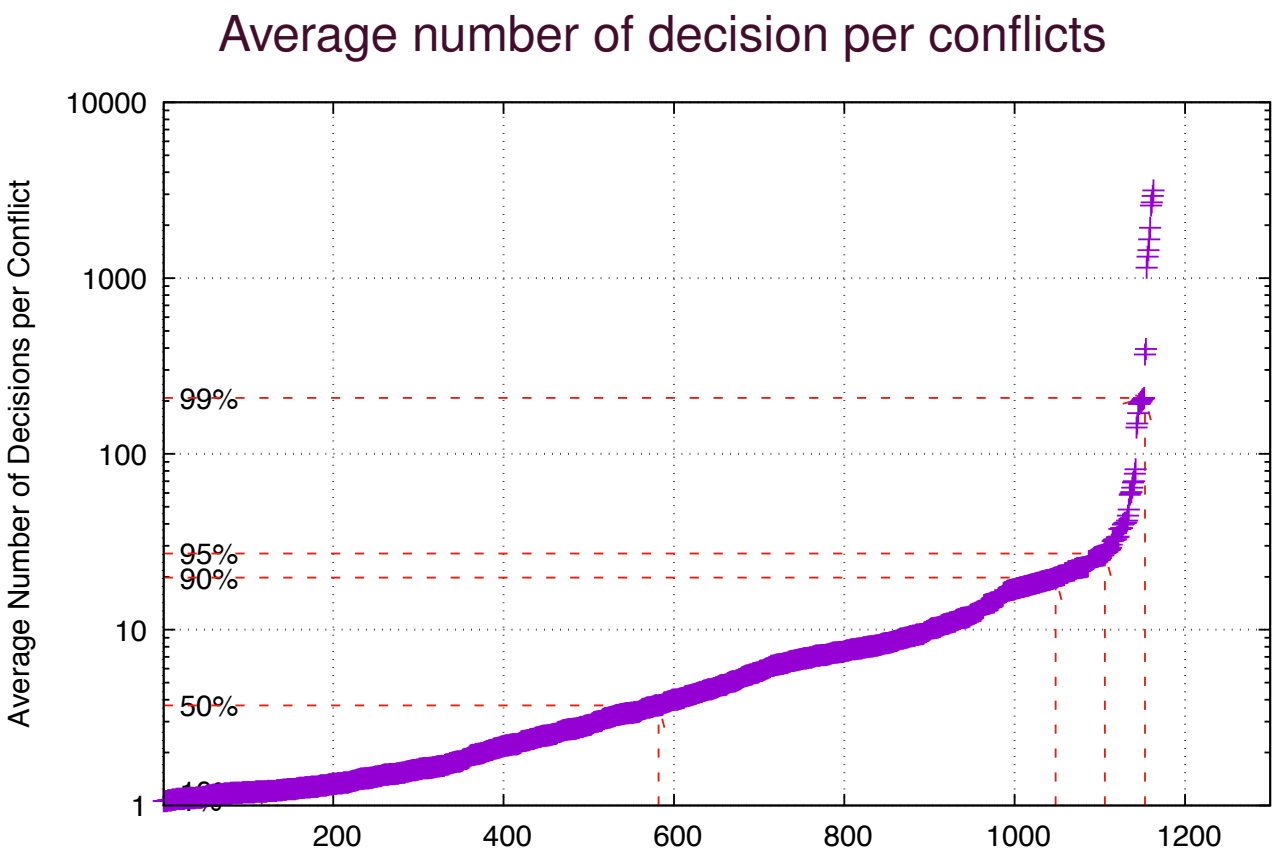
We try to understand CDCL solvers but all problems are distincts !



stats after 100,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

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

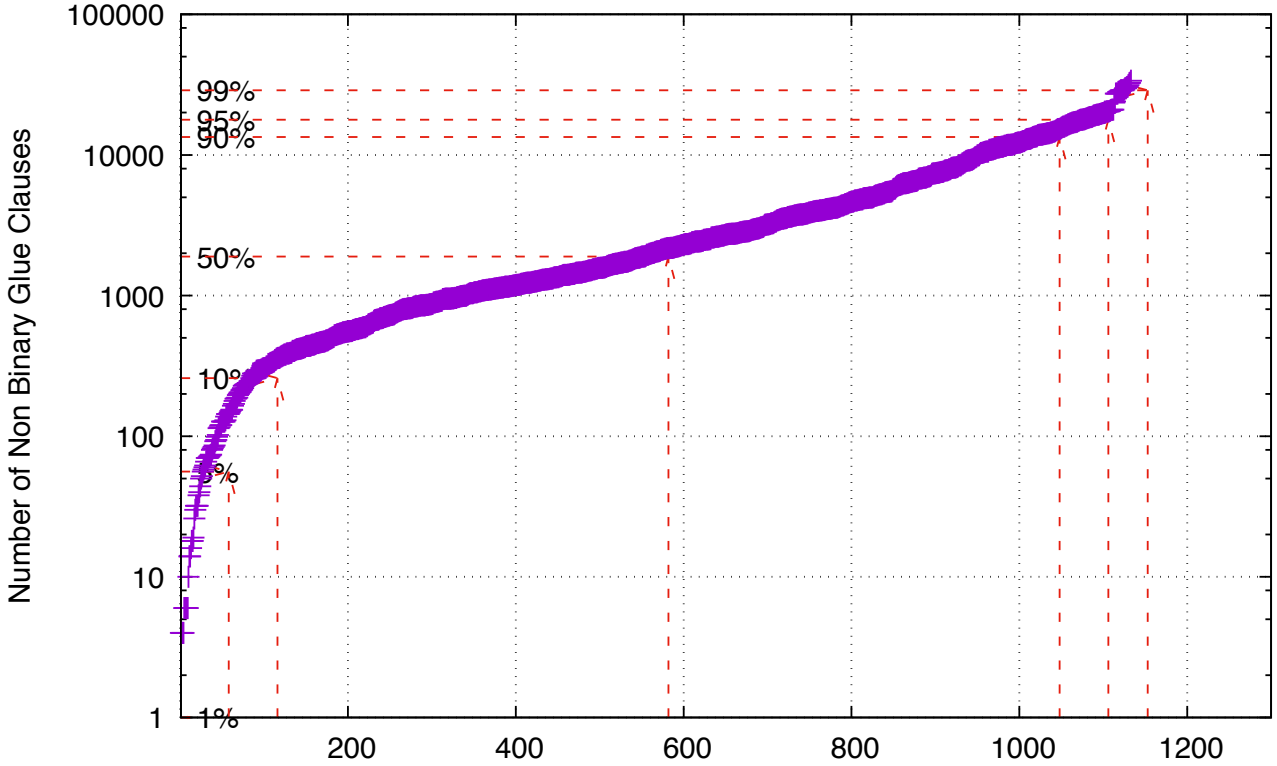


stats after 100,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

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

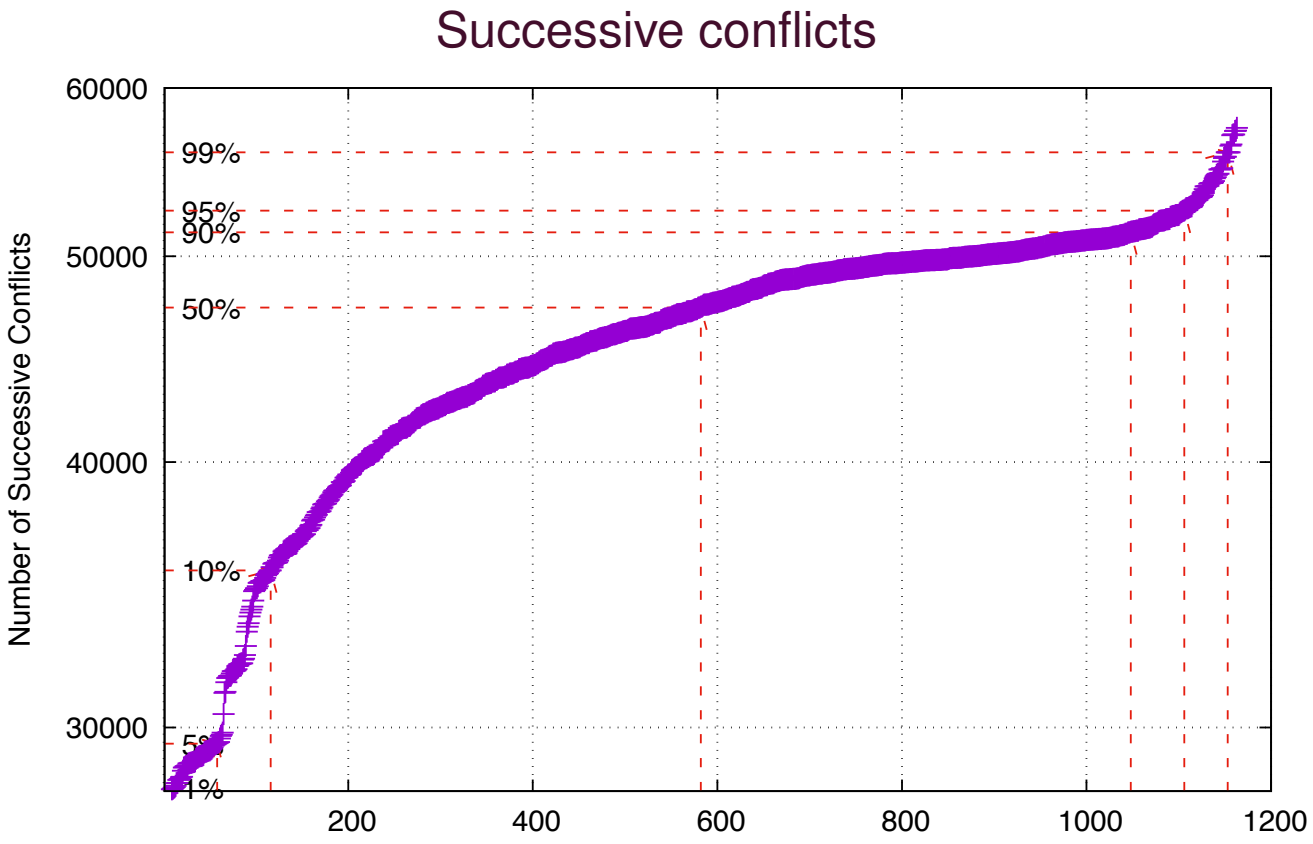
Average “True” (non binary) glue clauses



stats after 100,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

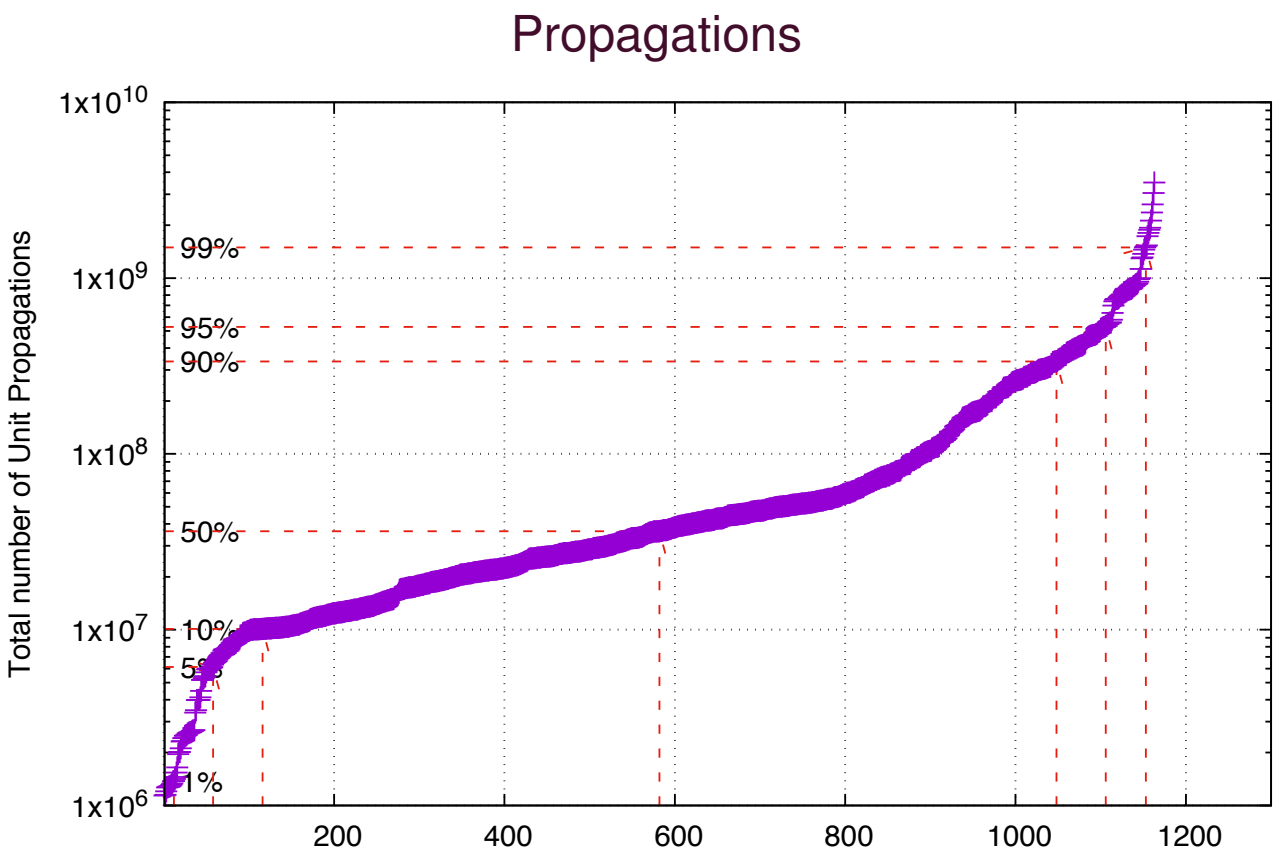
We try to understand CDCL solvers but all problems are distincts !



stats after 100,000 conflicts on 1164 “not easy” problems from all previous contests

Outliers everywhere !

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

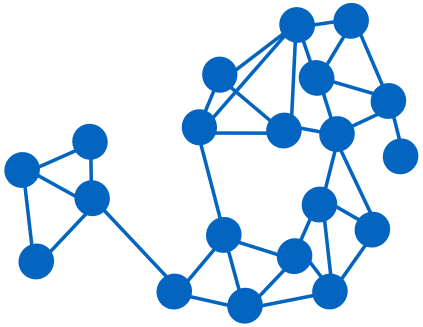


stats after 100,000 conflicts on 1164 “not easy” problems from all previous contests

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

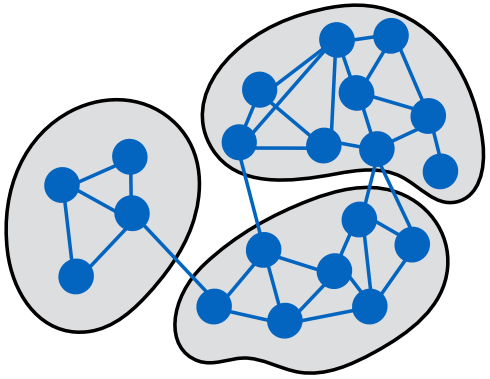


- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

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

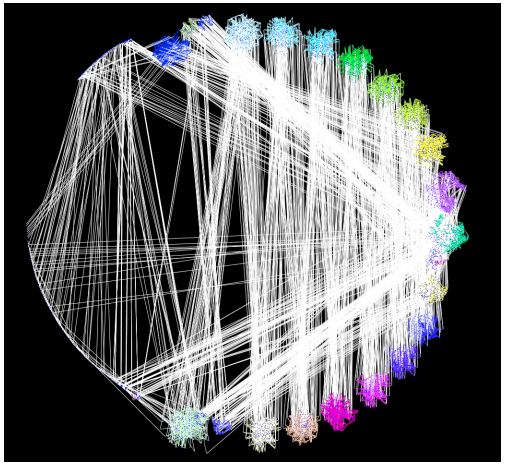


- Industrial instances have a high community structure
- Learning does not completely destroy communities
- Clauses with small LBD are (globally) inside the same community

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



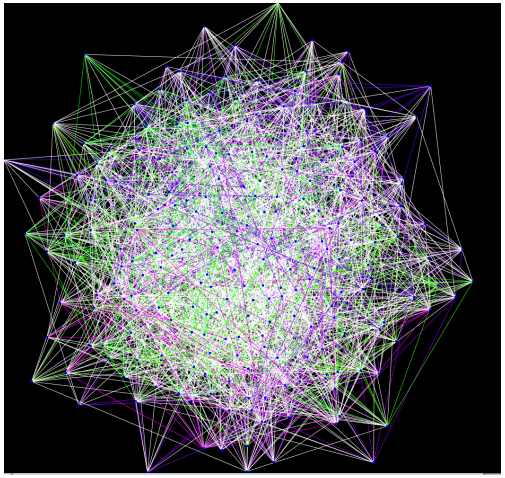
BMC instance

- 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

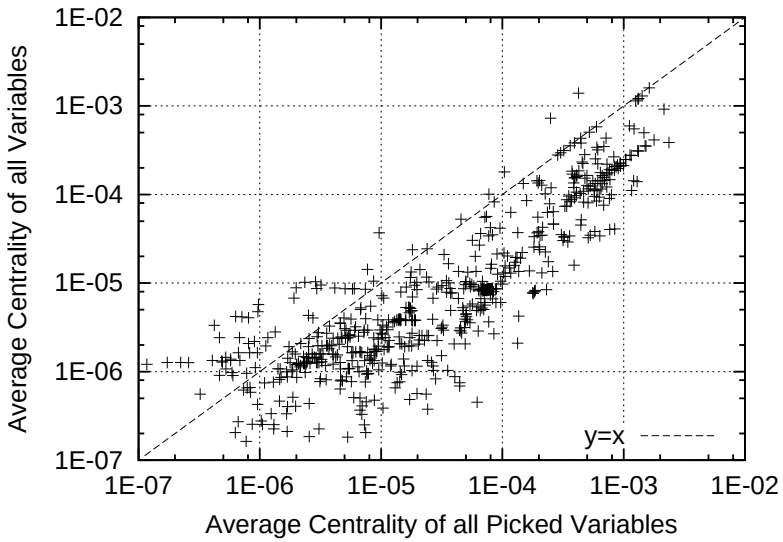
Centrality of variables

- 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

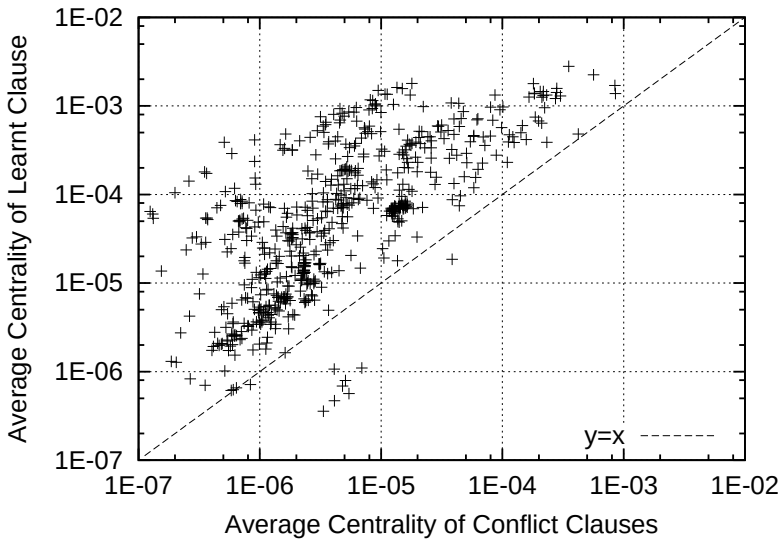
- Here again, a graph representation of a SAT instance
- Measure the centrality of variables
- Establish a link between
  - Centrality and VSIDS heuristic
  - Centrality and learning

- Decision variables are central !

Try to Explain Efficiency of CDCL Solvers

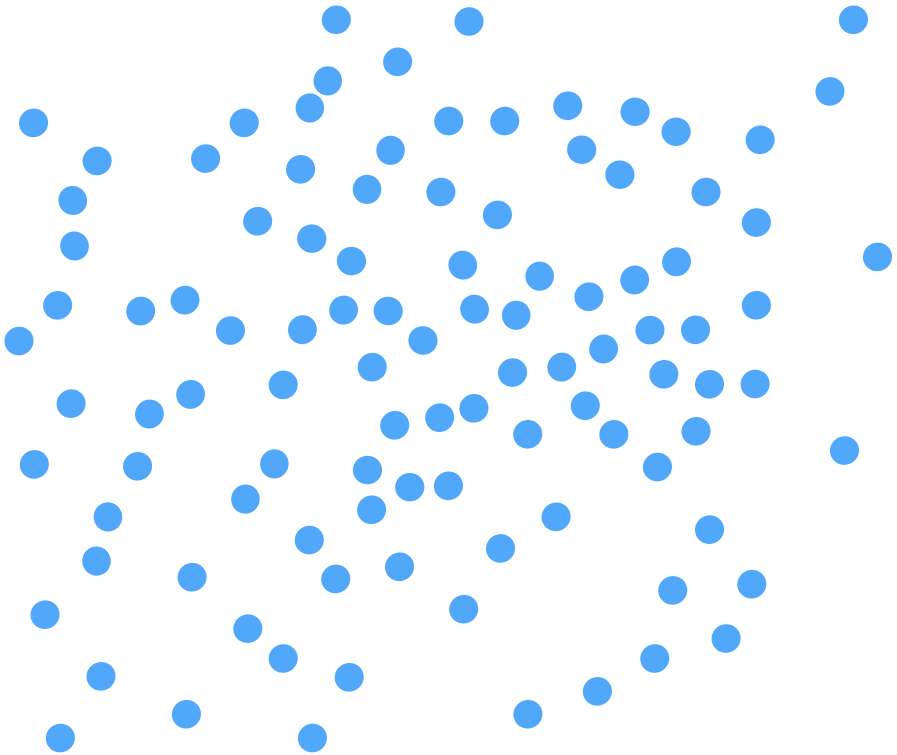
Centrality of variables

- 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

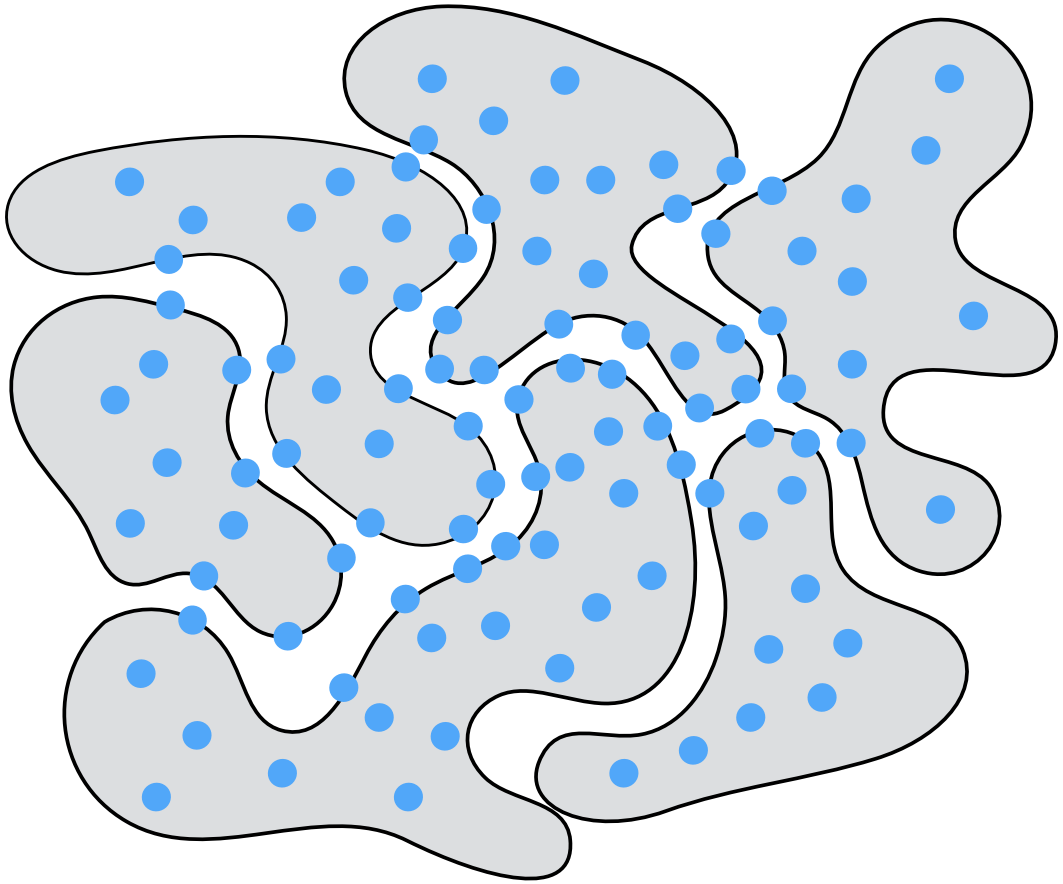
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

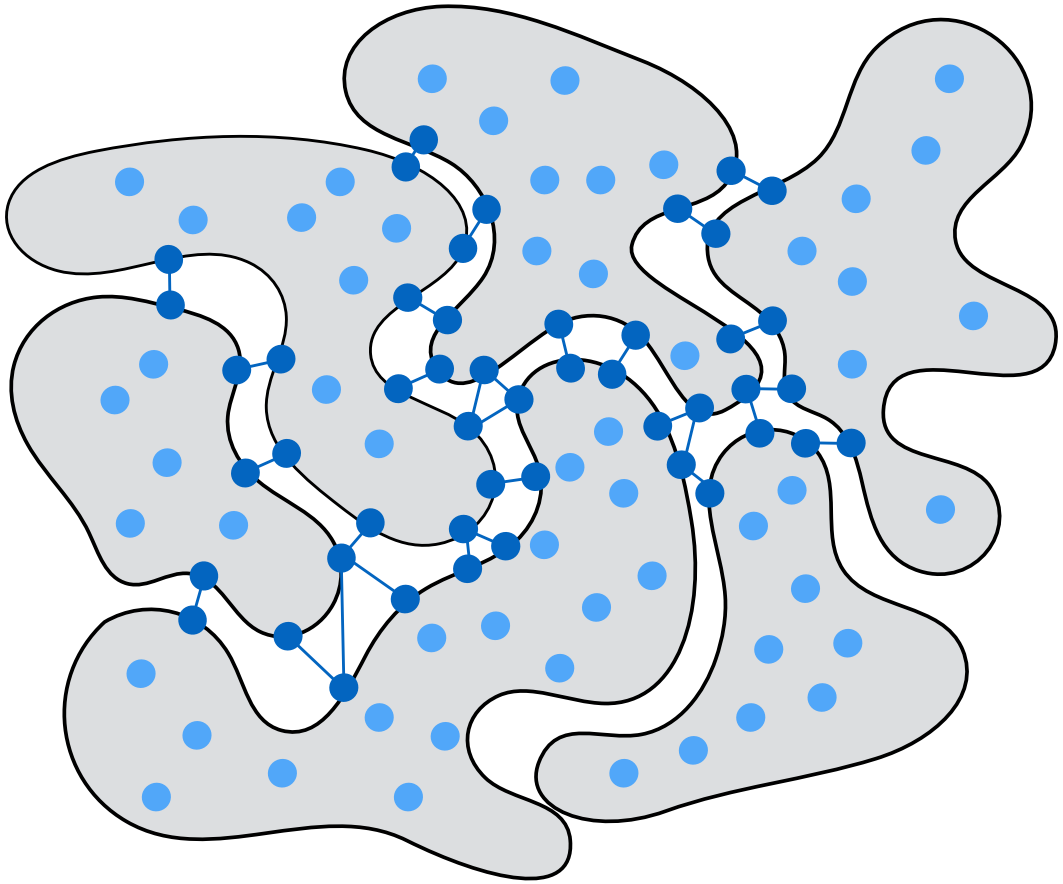
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

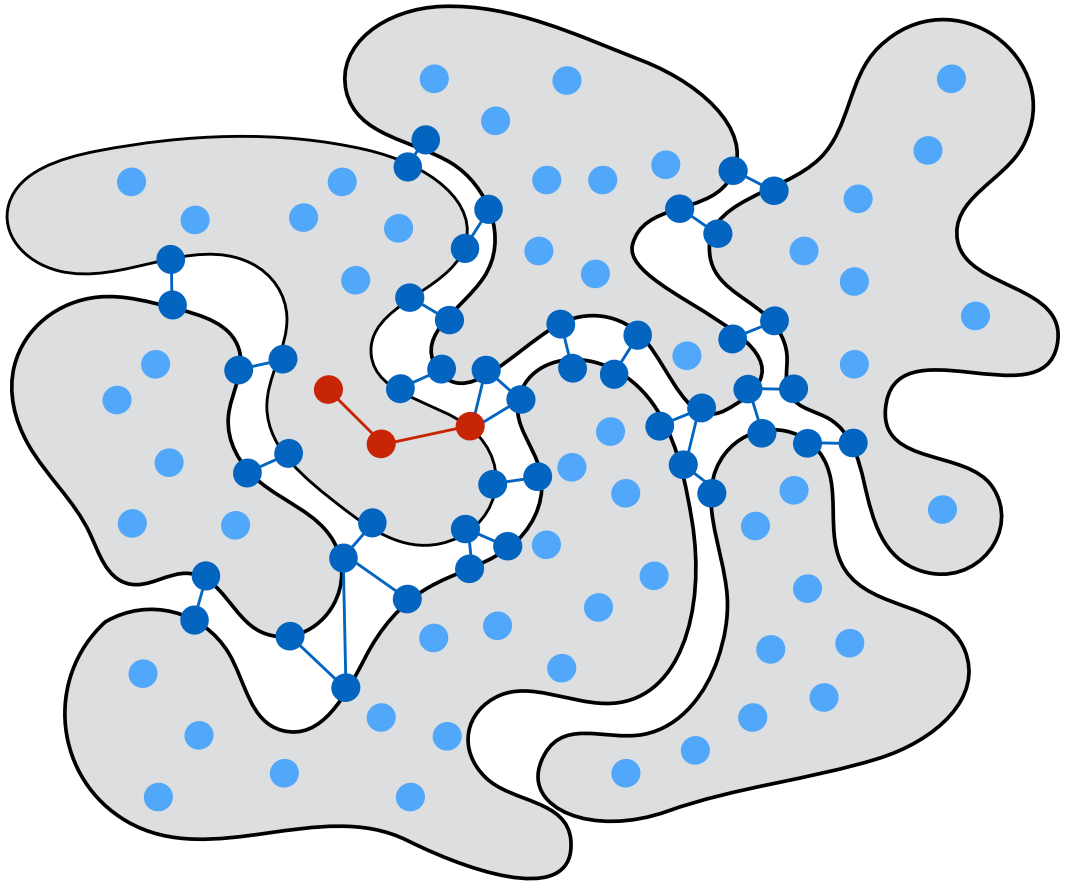
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

© Laurent Simon

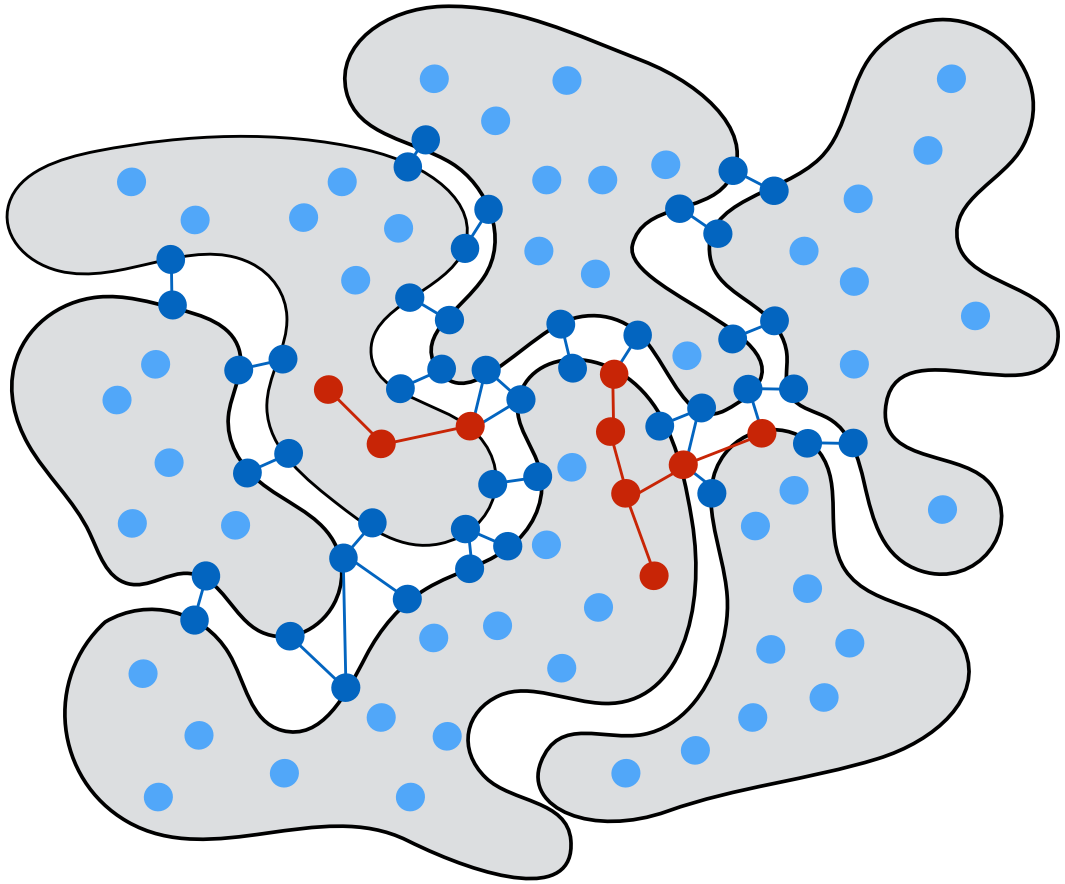
A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

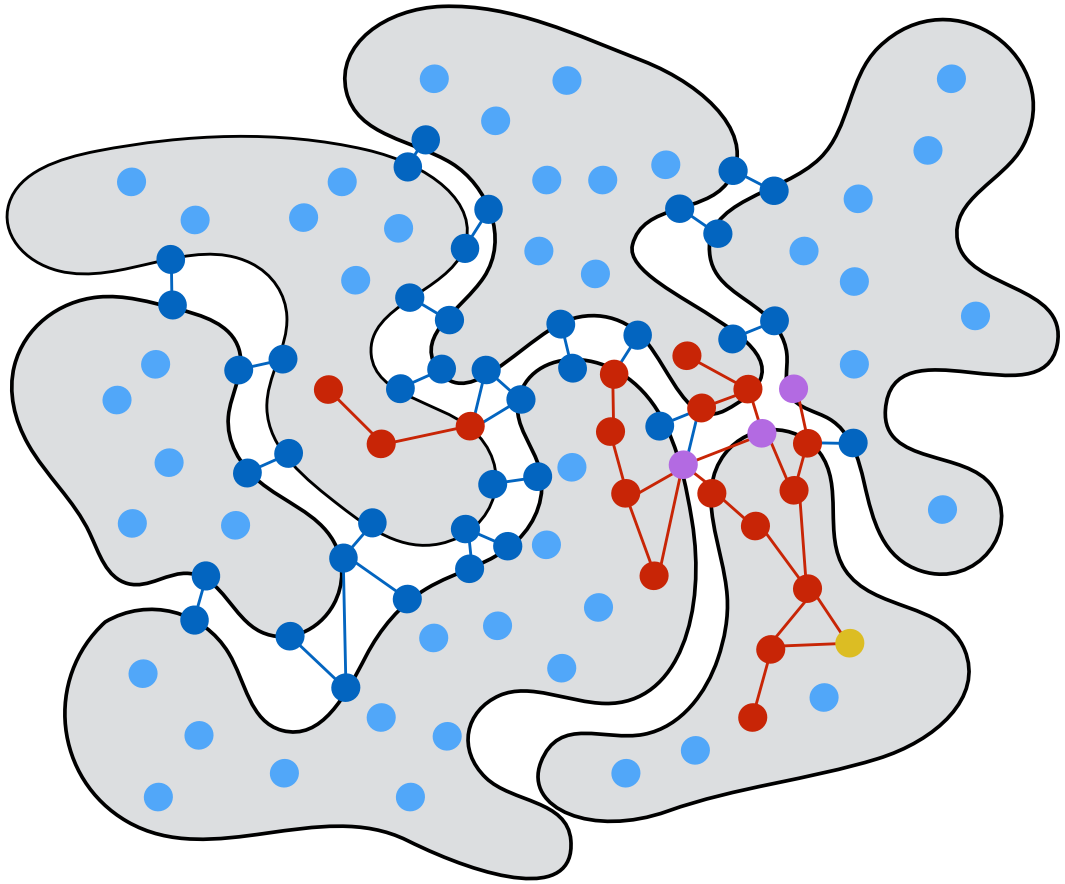
© Laurent Simon

A Possible Illustration of CDCL Solvers



InCluster Variable
Frontiere Variable
Propagated Variable
Conflict Variable
Learnt Clause Variable

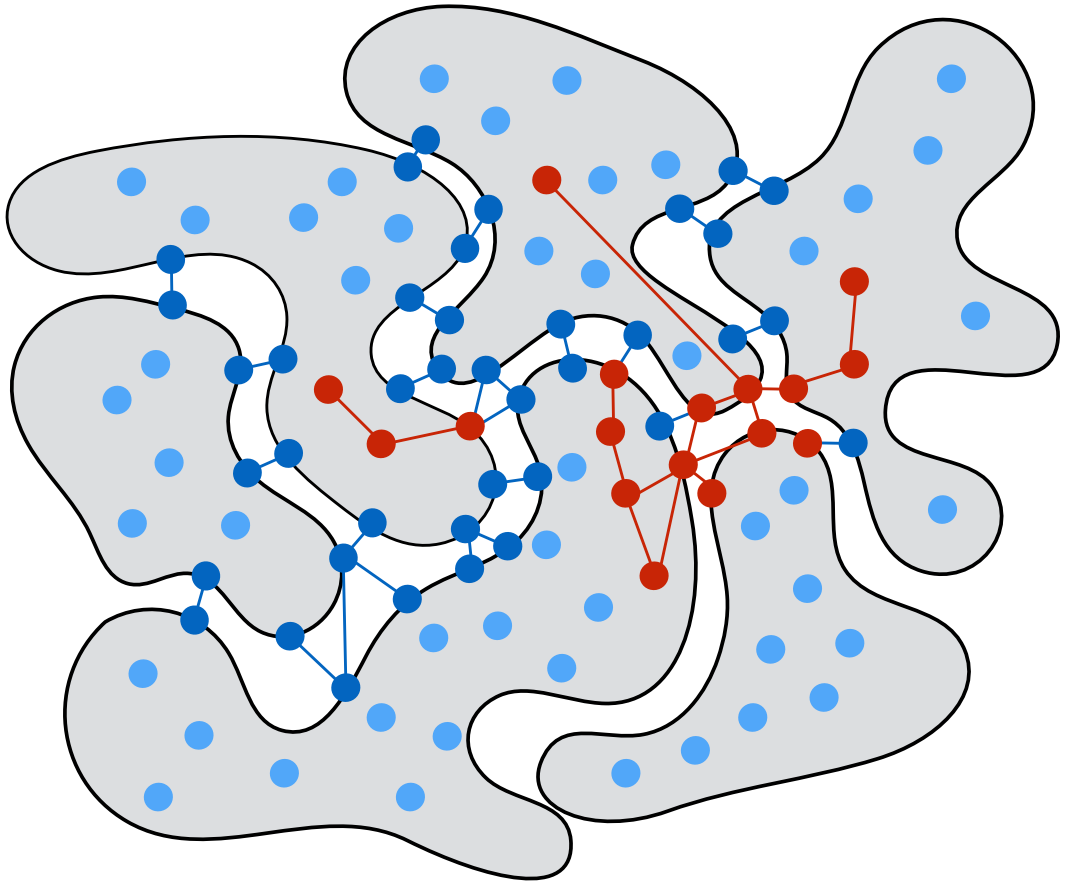
A Possible Illustration of CDCL Solvers



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

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

Exercise
Extract all models of a CNF