Logical reasoning and programming

SAT solving—CDCL and probabilistic methods

Karel Chvalovský

CIIRC CTU

Recap

We deal with formulae in conjunctive normal form (CNF)

$$(\cdots \vee \cdots \vee \cdots) \wedge \cdots \wedge (\cdots \vee \cdots \vee \cdots)$$

and we represent them using

$$\{\{\dots\}, \dots, \{\dots\}\}.$$

Our problem, given a set of clauses φ:

Is $\varphi \in \texttt{SAT}$?

DPLL algorithm

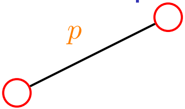
Require: A set of clauses φ
  function $\mathrm{DPLL}(\varphi)$
    while φ contains a unit clause $\{l\}$ **do** ▷ unit propagation
      delete clauses containing l from φ ▷ unit subsumption
      delete $\bar{l}$ from all clauses in φ ▷ unit resolution
    if $\Box \in \varphi$ **then return** `false` ▷ empty clause
    while φ contains a pure literal l **do**
      delete clauses containing l from φ
    if $\varphi = \emptyset$ **then return** `true` ▷ no clause
    else
      $l \leftarrow$ select a literal occurring in φ ▷ a choice of literal
      if $\mathrm{DPLL}(\varphi \cup \{\{l\}\})$ **then return** `true`
      else if $\mathrm{DPLL}(\varphi \cup \{\{\bar{l}\}\})$ **then return** `true`
      else return `false`

Example: DPLL (without pure literal elimination!)

$$
\begin{aligned}
c_1 &= \{p, q\} \\
c_2 &= \{q, r\} \\
c_3 &= \{\overline{p}, \overline{s}, t\} \\
c_4 &= \{\overline{p}, s, u\} \\
c_5 &= \{\overline{p}, \overline{t}, u\} \\
c_6 &= \{\overline{p}, s, \overline{u}\} \\
c_7 &= \{\overline{p}, \overline{t}, \overline{u}\}
\end{aligned}
$$

Clearly, q (a pure literal) and $\overline{p}$ (a pure literal after satisfying c_1 by q) satisfy clauses $c_1, \ldots, c_7$. The sole purpose of this example (a nonsensical run of DPLL) is to motivate CDCL.

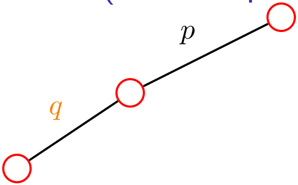
Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

For simplicity, we fix the order of choices to $p < q < r < s < t < u$ and always select a positive literal first, but any unselected literal can be chosen and in any order (positive/negative).

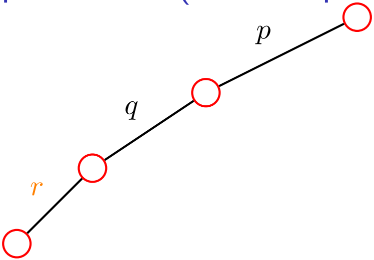
Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

For simplicity, we fix the order of choices to $p < q < r < s < t < u$ and always select a positive literal first, but any unselected literal can be chosen and in any order (positive/negative).

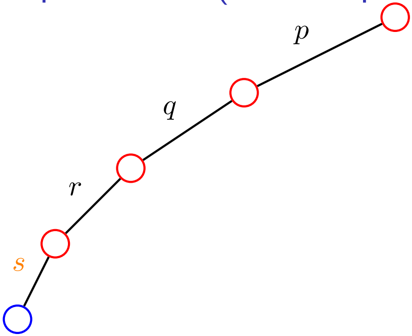
Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

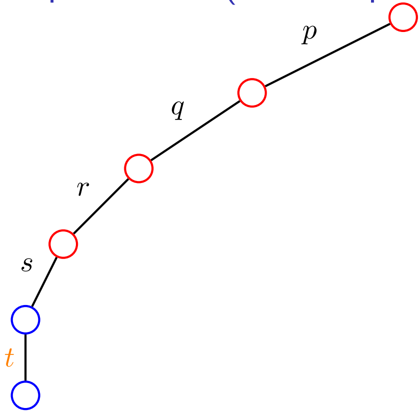
It makes no sense to select r here and it even violates the rule in DPLL, but it demonstrates a property of the algorithm. Or assume that there are also other clauses $c_8, \ldots$ to be satisfied.

Example: DPLL (without pure literal elimination!)



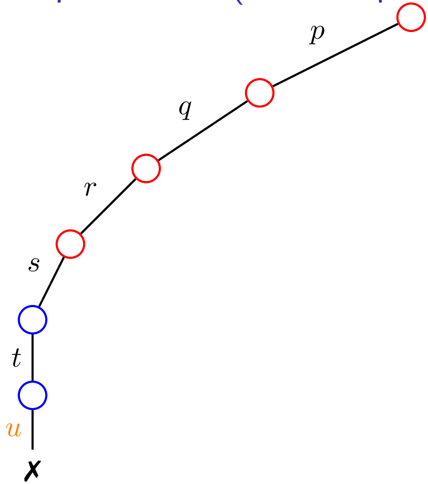
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



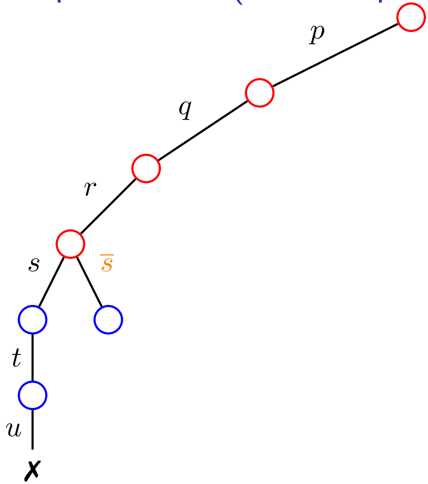
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



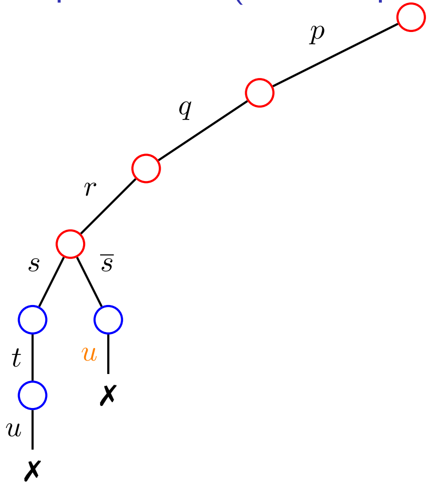
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



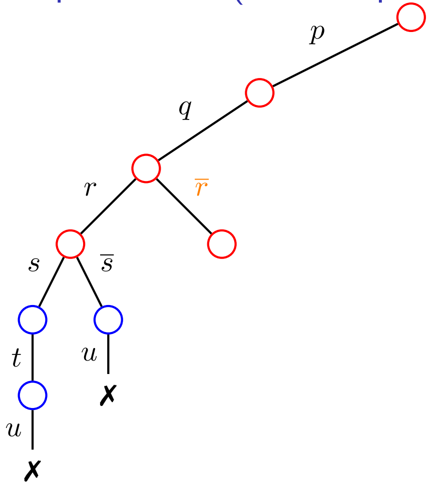
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



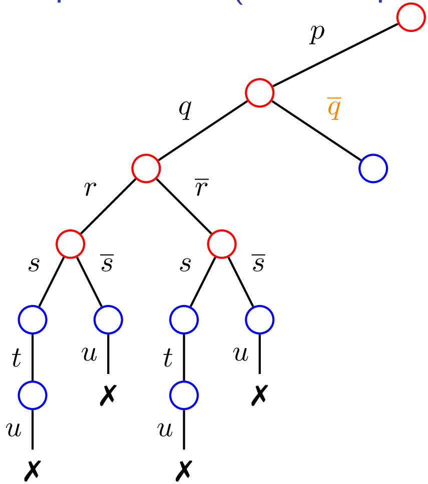
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



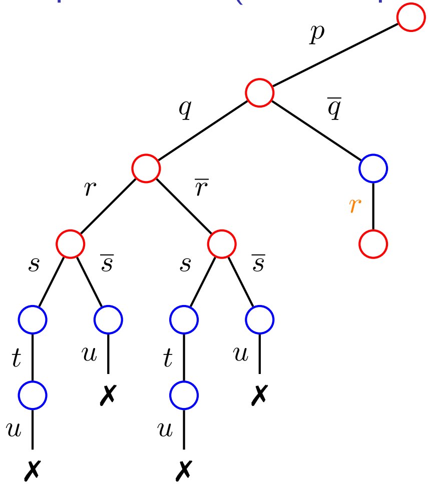
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



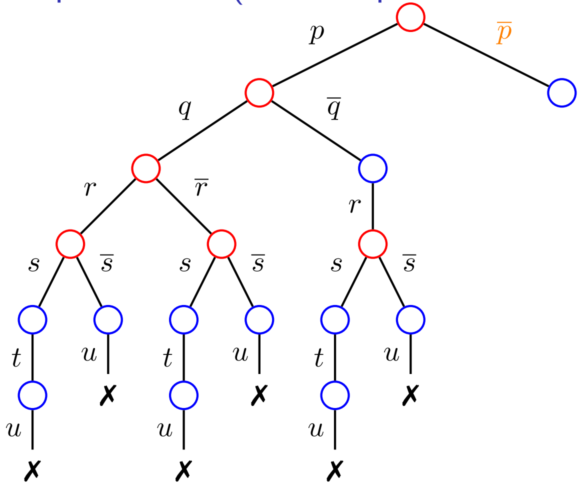
$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$

$c_2 = \{q, r\}$

$c_3 = \{\overline{p}, \overline{s}, t\}$

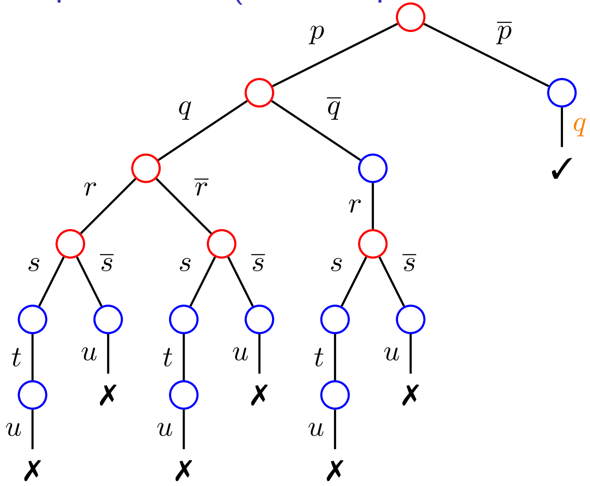
$c_4 = \{\overline{p}, s, u\}$

$c_5 = \{\overline{p}, \overline{t}, u\}$

$c_6 = \{\overline{p}, s, \overline{u}\}$

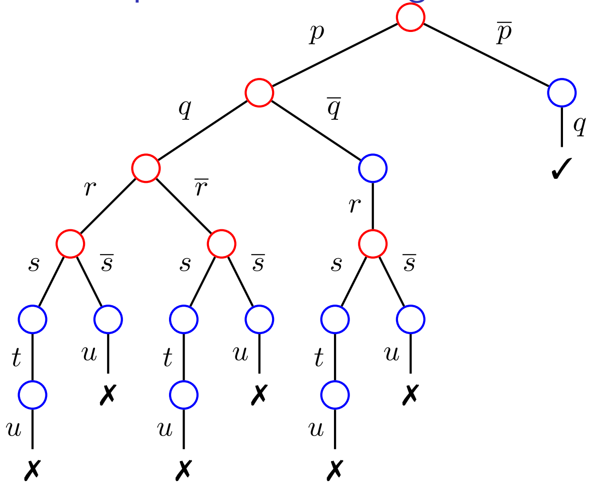
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Example: DPLL (without pure literal elimination!)



$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

How to improve backtracking in DPLL?

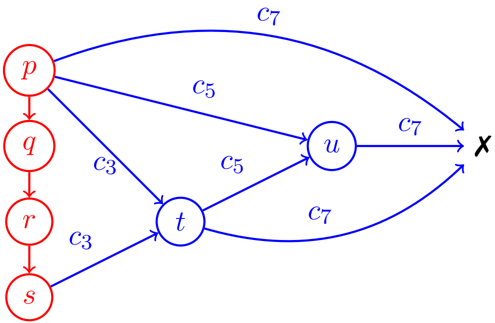


$$c_1 = \{p, q\}$$
$$c_2 = \{q, r\}$$
$$c_3 = \{\overline{p}, \overline{s}, t\}$$
$$c_4 = \{\overline{p}, s, u\}$$
$$c_5 = \{\overline{p}, \overline{t}, u\}$$
$$c_6 = \{\overline{p}, s, \overline{u}\}$$
$$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$$

Clearly, detected conflicts do not depend on q and r. Hence there is no need to check different assignments for them and we have a non-chronological backtracking.

Implication graph — analyzing conflicts

Red vertices are decision points and blue vertices are caused by unit propagations. Red edges show the direction of decisions and blue edges the reasons for unit propagations.

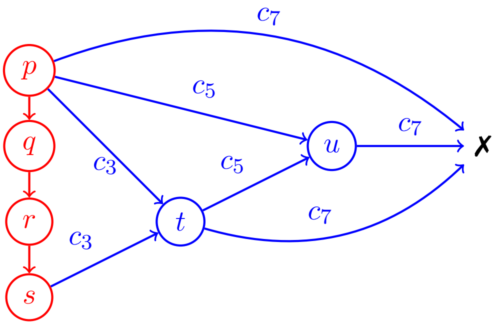


$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

Hence $(p \wedge s) \rightarrow \perp$ that is equivalent to $\{\overline{p}, \overline{s}\}$. We can learn this clause and add it to our set of clauses. This prevents us from visiting the same conflict in a different branch.

Implication graph — analyzing conflicts

Red vertices are decision points and blue vertices are caused by unit propagations. Red edges show the direction of decisions and blue edges the reasons for unit propagations.

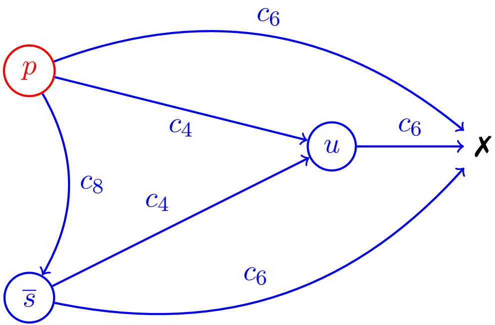


$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \bar{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \bar{t}, \overline{u}\}$

Moreover, $\{\overline{p}, \overline{s}\}$ is an *asserting clause*; it contains exactly one literal that depends on the last decision. Hence an asserting clause flips a literal on the last decision level. We learn only such clauses.

Implication graph — analyzing conflicts

We can also analyze the second conflict now.

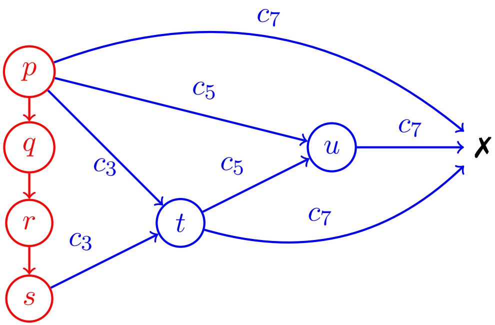


$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$
$c_8 = \{\overline{p}, \overline{s}\}$

Hence we learn $c_9 = \{\overline{p}\}$.

Implication graph — various cuts

It was possible to learn a different clause.

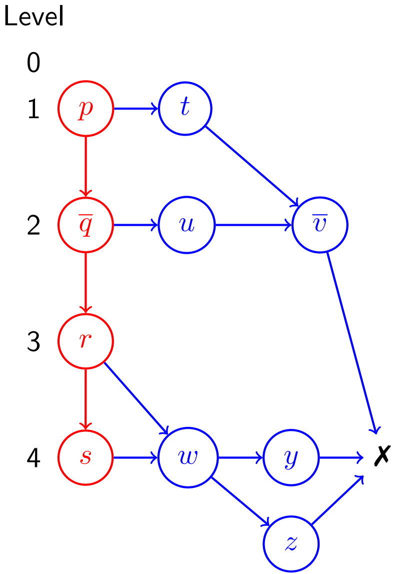


$c_1 = \{p, q\}$
$c_2 = \{q, r\}$
$c_3 = \{\overline{p}, \overline{s}, t\}$
$c_4 = \{\overline{p}, s, u\}$
$c_5 = \{\overline{p}, \overline{t}, u\}$
$c_6 = \{\overline{p}, s, \overline{u}\}$
$c_7 = \{\overline{p}, \overline{t}, \overline{u}\}$

We usually prefer to learn $\{\overline{p}, \overline{t}\}$ instead of $\{\overline{p}, \overline{s}\}$. Because t is so called dominator—all paths from s to the conflict go through t.

We call such dominators *unique implication points* (UIP) and a popular strategy is to learn the first UIP (the one closest to the conflict) on the path to the last decision point. Why? They tend to be shorter.

Implication graph — various decision levels



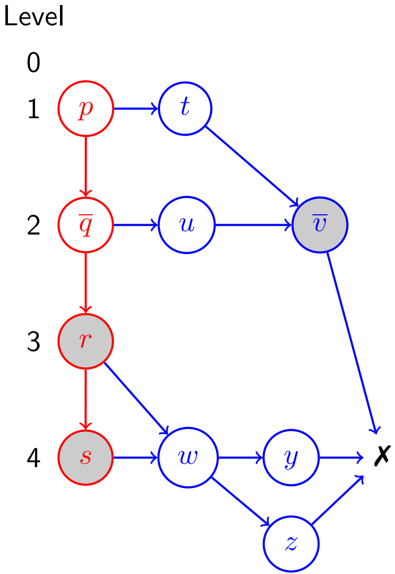
We want

- a literal assigned at the last level,
- literals involved assigned at previous levels.

Not necessarily decision literals!

Conflict analysis—we go from the conflict to the last decision literal and add involved literals from previous levels. Hence it is fast!

Implication graph — various decision levels



We want

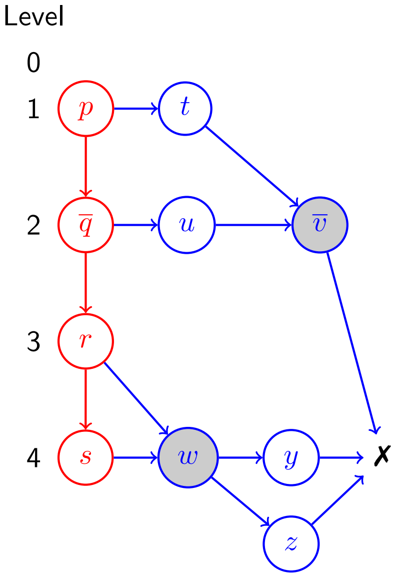
- a literal assigned at the last level,
- literals involved assigned at previous levels.

Not necessarily decision literals!

Here one option is

$$\{\overline{s}, \overline{r}, v\}.$$

Implication graph — various decision levels



We want

- a literal assigned at the last level,
- literals involved assigned at previous levels.

Not necessarily decision literals!

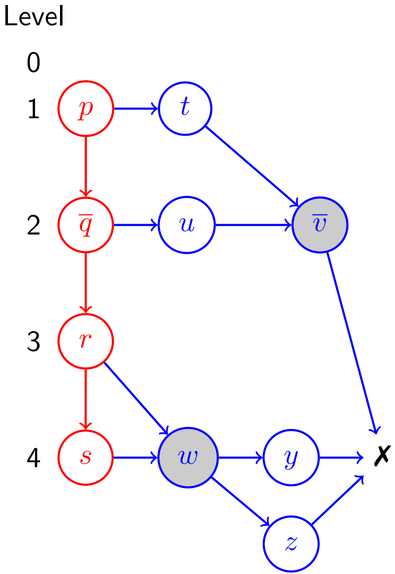
Here one option is

$$\{\overline{s}, \overline{r}, v\}.$$

However, the first UIP gives

$$\{\overline{w}, v\}.$$

Implication graph — various decision levels



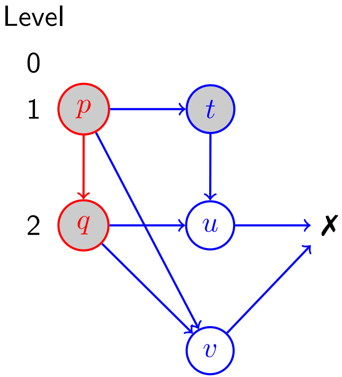
We want

- a literal assigned at the last level,
- literals involved assigned at previous levels.

Not necessarily decision literals!

We backtrack to the decision level 2 (and obtain $\overline{w}$), because that is the maximal decision level in the learned clause when we ignore the literal from the last decision level ($\overline{w}$ here).

Learned clause minimization (local)



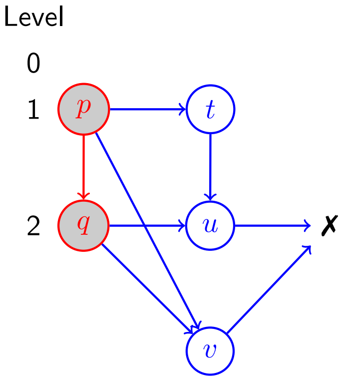
Here we learn

$$\{\overline{q}, \overline{p}, \overline{t}\}.$$

However, we get t using

$$\{\overline{p}, t\}.$$

Learned clause minimization (local)



Here we learn

$$\{\overline{q}, \overline{p}, \overline{t}\}.$$

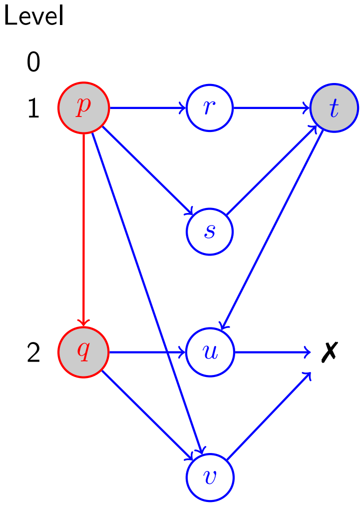
However, we get t using

$$\{\overline{p}, t\}.$$

Hence by self-subsumption resolution we obtain

$$\{\overline{q}, \overline{p}\}.$$

Learned clause minimization (recursive)



Here we learn

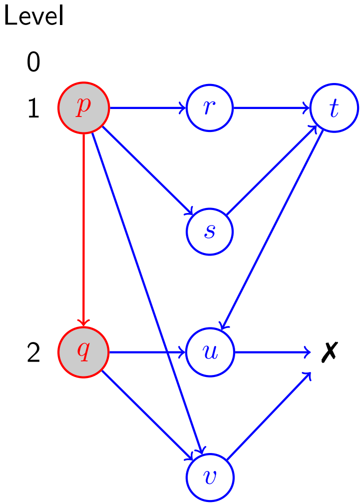
$$\{\overline{q}, \overline{p}, \overline{t}\}.$$

and the previous approach fails here; we get a longer clause using

$$\{\overline{r}, \overline{s}, t\},$$

which gives us t.

Learned clause minimization (recursive)



Here we learn

$$\{\overline{q}, \overline{p}, \overline{t}\}.$$

and the previous approach fails here; we get a longer clause using

$$\{\overline{r}, \overline{s}, t\},$$

which gives us t.

Hence a recursive minimization, which is more time-consuming, is necessary to get

$$\{\overline{q}, \overline{p}\}$$

taking into account also clauses $\{\overline{p}, r\}$ and $\{\overline{p}, s\}$.

Conflict-Driven Clause Learning (CDCL)

It is the DPLL algorithm with non-chronological backtracking, called back jumping, and clause learning. However, CDCL with all the restarts and the deletions of learned clauses has little in common with a systematic search done by DPLL.

Restarts

It is useful to restart a CDCL solver from time to time. We forget all assignments but keep the learned clauses.

Delete learned clauses

It is necessary to delete some learned clauses to avoid space problems and hence we try to keep only the most useful clauses.

SAT/UNSAT modes

Modern solvers use different modes for SAT/UNSAT problems, or alternate these modes during their run.

Preprocessing

We want to obtain an equisatisfiable problem that is "simpler".

There are many techniques

- unit propagations,
- pure literal eliminations,
- subsumptions,...

Bounded Variable Elimination (BVE)[1]

Loosely speaking, we eliminate a variable as in Davis–Putnam only when it does not increase the number of clauses (this can be relaxed over time). Combined with tautology elimination and subsumptions.

Inprocessing

Basically all state-of-the-art solvers interleave search with preprocessing.

[1] There exists also Bounded Variable Addition (BVA).

Decision heuristics

How to select a literal? Many approaches, but it has to be fast.

Historically

Based on the number of occurrences of variables in unsatisfied clauses. Many variants, for example,

- considered only the shortest unsatisfied clauses,
- weight their occurrences (Jeroslow–Wang)

$$w(l) = \sum_{c \in \varphi \text{ such that } l \in c} 2^{-|c|}$$

We can compute it at the beginning or dynamically, however, that is expensive to do, cf. watched literals.

Why do we prefer short clauses?

Decision heuristics — modern

Focus heuristics

In CDCL we try to find small unsatisfiable subsets and hence we prefer variables involved in recent conflicts.

Modern solvers usually use a variant of VSIDS (Variable State Independent Decaying Sum).

- Initialize with the number of occurrences of a variable in φ,
- if a conflict clause c is learned, then the score of all variables in c is increased, and
- we periodically divide our scores by a constant to prioritize recently learned clauses.

Global heuristics

We look-ahead on a literal l. It means that we assume l, then we apply unit propagations and check clauses that are shortened by this assignment, but not completely satisfied. We prefer literals that produce shorter clauses. We also learn if possible. Good for random k-SAT.

Decision heuristics — value

We have selected a variable, but what value (positive/negative) should we try first? It is also called phase picking and it is especially important for satisfiable instances.

Historically

- based on the number of occurrences of variables in unsatisfied clauses; many variants
- a version of MiniSAT always sets literals to false

Phase saving

We do not concentrate directly on clauses, but instead we cache the behavior of variables during propagations and backtracking; we want to reach similar regions of the search space. Also very useful in combination with rapid restarts; we keep exploring the same region of the search space.

Parallel solving

SAT solving is difficult to parallelize. Moreover, our data structures, e.g. watched literals, make it even harder.

Cube and conquer (look-ahead and CDLC)

We generate many partial assignments, e.g., by a breath-first search with a limited maximal depth, and try to solve them.

Good for hard combinatorial problems, e.g., the Boolean triples problem.

Portfolio approach

We run multiple solvers (usually the same one) with different settings on the same formula. We share clauses, which is especially important for unsatisfiable instances, among solvers. The main problems are how to diversify our portfolio and share clauses (which clauses, how many of them, when, . . .).

It works very well on large problems that are easy to solve.

Probabilistic algorithms — stochastic local search

We start with a random complete valuation and try to minimize the number of unsatisfied clauses by flipping variables.

These methods are incomplete and it is an open problem how to use these techniques for showing unsatisfiability.

GSAT

Require: A set of clauses φ
  function $\mathrm{GSAT}(\varphi)$
    for $i \in (1, MAXITERS)$ **do**
      $v \leftarrow$ a random valuation on φ
      for $j \in (1, MAXFLIPS)$ **do**
        if $v \models \varphi$ **then return** v
        else minimize #unsat clauses by flipping a variable
    return None

Many extensions and variants, the most famous one is WalkSAT. You can try some of them in UBCSAT.

WalkSAT

We try to avoid local minima by combining the greedy moves of GSAT with random walk moves.

- Select randomly an unsatisfied clause c.
  - If by flipping a variable x occurring in c no satisfied clause becomes unsatisfied, then flip x. ("freebie" move)
  - Otherwise with a probability
    - p flip a random variable x in c ("random walk" move),
    - $(1-p)$ perform a GSAT step ("greedy" move) on variables from c; flip the best variable $x \in c$.

For details see Walksat Home Page. It is efficient on random k-SAT. Also historically good for planning and circuit design problems.

probSAT

A generalization of WalkSAT that calculates the probability distribution for the potential flip variables. It works also on some hard non-random problems. For details, see here.

CDCL or/and stochastic local search

On some instances stochastic local search methods work very well, you can try UBCSAT. But the previous methods, based on CDCL, usually outperform them and, moreover, are able to show that a problem is UNSAT.

CDCL or/and stochastic local search

On some instances stochastic local search methods work very well, you can try UBCSAT. But the previous methods, based on CDCL, usually outperform them and, moreover, are able to show that a problem is UNSAT.

However, it is possible to combine CDCL with a local search and many modern solvers take advantage of that

- for example, we can use a local search to produce a long partial assignment (trail) and then use this knowledge when we decide the values of variables (phase picking).

The Nobel Prize in Physics 2021

Giorgio Parisi (Prize share: $1/2$)

For the discovery of the interplay of disorder and fluctuations in physical systems from atomic to planetary scales.

The Nobel Prize in Physics 2021 and SAT

Giorgio Parisi (Prize share: $1/2$)

For the discovery of the interplay of disorder and fluctuations in physical systems from atomic to planetary scales.

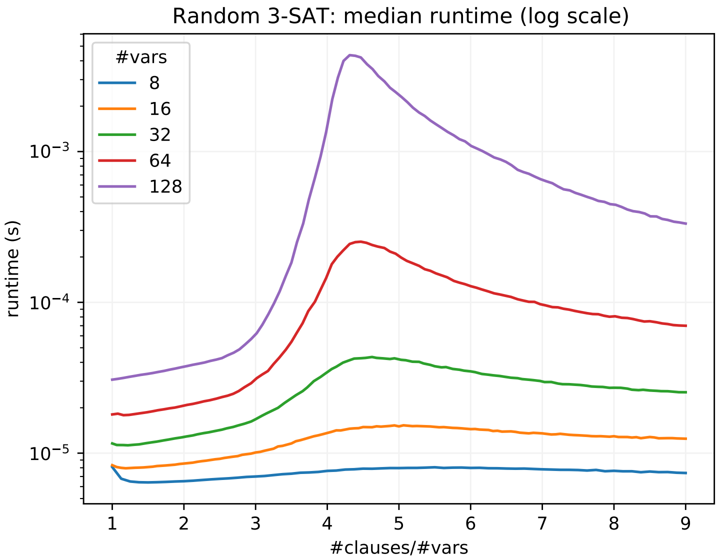
Analytic and Algorithmic Solution of Random Satisfiability Problems

M. Mézard,[1] G. Parisi,[1,2] R. Zecchina[1,3]*

We study the satisfiability of random Boolean expressions built from many clauses with *K* variables per clause (K-satisfiability). Expressions with a ratio α of clauses to variables less than a threshold α_c are almost always satisfiable, whereas those with a ratio above this threshold are almost always unsatisfiable. We show the existence of an intermediate phase below α_c, where the proliferation of metastable states is responsible for the onset of complexity in search algorithms. We introduce a class of optimization algorithms that can deal with these metastable states; one such algorithm has been tested successfully on the largest existing benchmark of K-satisfiability.

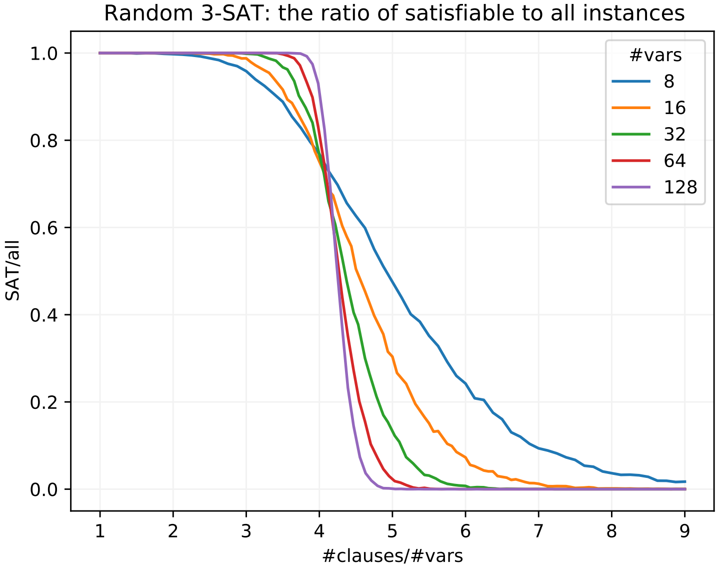
Phase transition for random k-SAT

(literals are selected randomly, each clause has length exactly k)



Phase transition for random k-SAT

(literals are selected randomly, each clause has length exactly k)



Bibliography I

- Biere, Armin, Marijn Heule, et al., eds. (2021). *Handbook of Satisfiability*. 2nd. Vol. 336. Frontiers in Artificial Intelligence and Applications. Washington: IOS Press. ISBN: 978-1-64368-161-0.
- Biere, Armin, Marijn J. H. Heule, et al., eds. (Feb. 2009). *Handbook of Satisfiability*. Vol. 185. Frontiers in Artificial Intelligence and Applications. IOS Press, p. 980. ISBN: 978-1-58603-929-5.