SAT + SMT

December 10-12, 2021

IARCS

An IARCS Winter School

A Tutorial on SAT Solving

Shaowei Cai

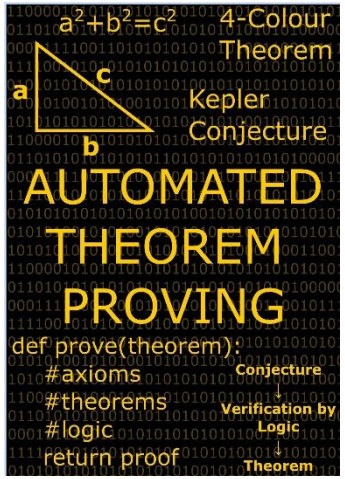
Institute of Software, Chinese Academy of Sciences

2021.12.12

Two ways of solving with computer

Problems can be stated formally

- the user states the problem and the computer solves it by solving the mathematical problem.



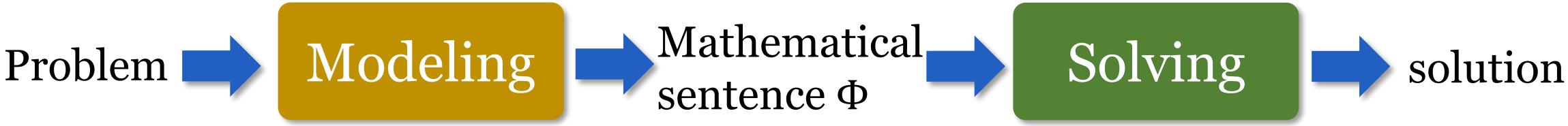
Typical Scenario： theorem proving

Problems cannot be stated formally

- the user provides examples and the computer learns to solve it.

Typical Scenario： face recognition

Constraint Solving



Boolean algebra （SAT）
logical structure
$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

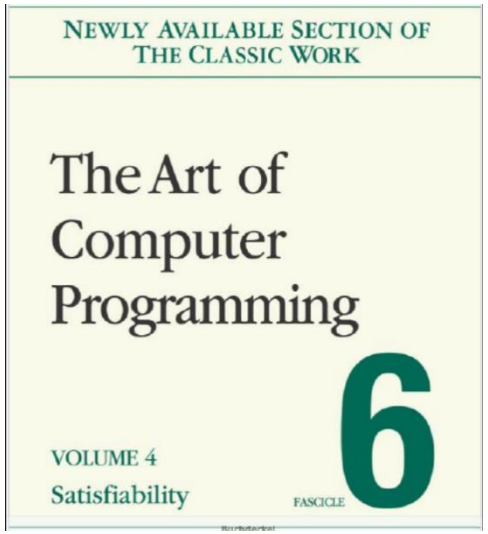
Constraint Programming（CSP）
various constraints
$\psi = AllDifferent(x_i, x_j) \wedge |x_i - x_j| \neq |i - j|$

Mathematical Progamming（MP）
Numerical constraints
$y^2 + 2xy < 100$
$x - y < 30$
$x \leq 60$

Restricted first order logic（SMT）
Logical structure + background theory
$\Phi = ((y^2 + 2xy < 100) \vee ((f(y) < 30) \rightarrow \neg(x - y < 30) \wedge (x \leq 60))$

Hello, SAT!

- The first NP-Complete problem [Cook, 1971]
- Conceptually simple
- Many theoretical results
- Many important applications
- Open source benchmarks and solvers
- Annual competition



The SAT problem is evidently a killer app, because it is key to the solution of so many other problems. SAT-solving techniques are among computer science's best success stories so far, and these volumes tell that fascinating tale in the words of the leading SAT experts.

Donald Knuth

. . . Clearly, efficient SAT solving is a key technology for 21st century computer science. I expect this collection of papers on all theoretical and practical aspects of SAT solving will be extremely useful to both students and researchers and will lead to many further advances in the field.

Edmund Clarke

Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search
- Hybrid SAT Solving

SAT

Propositional Satisfiability (SAT): Given a propositional formula φ, test whether there is an assignment to the variables that makes φ true.

e.g., a CNF formula

$$\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$$

- A core problem in computer science and a basic problem in logic
- Powerful SAT solvers widely used in industry and science
  - To find a certain object (a combination, a matching, a bug, a path…)
  - To prove a theorem (ϕ is tautology $\leftrightarrow$ $\neg\phi$ is UNSAT)

SAT

•SAT solvers usually adopt the Conjunctive Normal Form (CNF):

e.g., $\varphi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (x_2 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_3 \vee x_4)$

Every propositional formulas can be converted into CNF efficiently.

• Boolean variables: $x_1, x_2, \ldots$

• A literal is a Boolean variable x (positive literal) or its negation $\neg x$ (negative literal)

• A clause is a disjunction ($\vee$) of literals

$x_2 \vee x_3$,

$\neg x_1 \vee \neg x_3 \vee x_4$

• A Conjunctive Normal Form(CNF) formula is a conjunction ($\wedge$) of clauses.

SAT Basis

- A truth assignment φ assigns a truth value (True or False) to each Boolean variable x.
  - φ satisfies a clause if it satisfies at least one of its literals.
  - φ satisfies a CNF formula if it satisfies all of its clauses.

$$\{x_1 \mapsto 1,\ x_2 \mapsto 0,\ x_4 \mapsto 1\}\ ,$$

$$\begin{array}{ll} (x_1 \vee x_3 \vee \neg x_4) & \text{is satisfied,} \\ (\neg x_1 \vee x_2) & \text{is conflicting,} \\ (\neg x_1 \vee \neg x_4 \vee x_3) & \text{is unit,} \\ (\neg x_1 \vee x_3 \vee x_5) & \text{is unresolved.} \end{array}$$

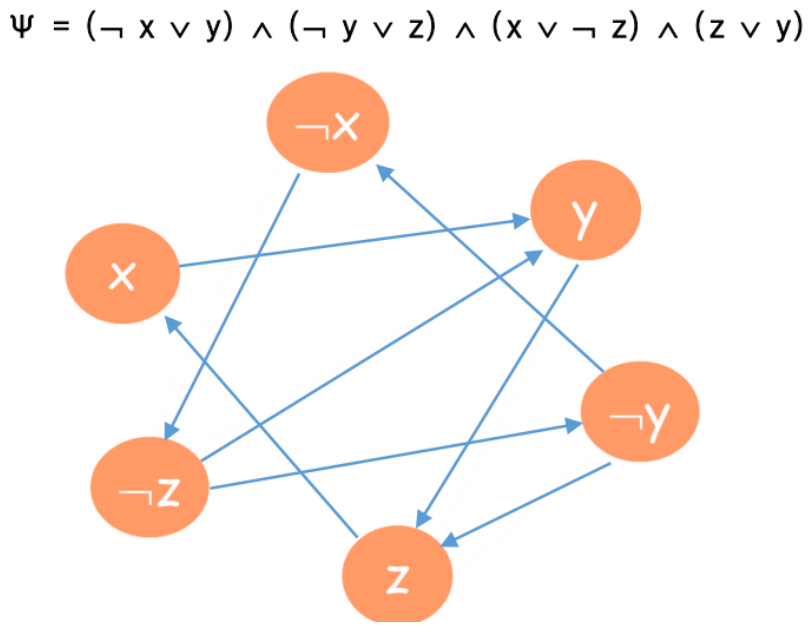
Tractable and intractable classes

- SAT is generally NP complete
  - some classes of the problem can be solved within polynomial time (2SAT, Horn-SAT).
  - Some classes are NP complete (3-SAT)

- Solving 2-SAT problem is easy

- For each clause $l \vee k$, produce two edges $\neg l \rightarrow k, \neg k \rightarrow l$

Check whether there a variable x, such that there is loop between x *and* $\neg x$.



Dichotomy theorem of SAT

- Are there any criteria that would allow us to distinguish between tractable and intractable classes?

- A remarkable result was obtained by Schaefer in 1978.

- Loosely speaking, Schaefer's dichotomy theorem states that there are only two possible cases: **SAT(C) is either in P or NP-complete**.
  - He considered Boolean constraint satisfaction problems SAT(C) where C is a set of constraints.
  - Each set C determines a class of formulas such as 2-CNF, 3-CNF, Horn formulas, etc.

Dichotomy theorem of SAT

- Are there any criteria that would allow us to distinguish between tractable and intractable classes?

Theorem (Schaefer, STOC 1978)

Given a Boolean constraint set C, SAT(C) is in P if C satisfies one condition below, and otherwise it is NP-complete:

- 1. C is 0-valid (1-valid);
- 2. C is weakly positive (weakly negative); // Horn, dual-Horn
- 3. C is bijunctive; //2-SAT
- 4. C is affine. //A system of linear equations

A CNF formula is Horn (dul-Horn, resp.) if every clause in this formula has at most one positive (negative, resp.) literal

Hardness Assumptions

For 3-SAT, the best known algorithms run in exponential time in n, which motivates the hardness assumption that better algorithms do not exist.

- Exponential Time Hypothesis (ETH)
  - 3-SAT cannot be solved in $2^{o(n)}$ time,

- Strong Exponential Time Hypothesis (SETH).
  - nonexistence of algorithms with running time bound $O^*(c^n)$ for some constant c < 2 for k-SAT with large k.

Upper bounds for 3-SAT

- Randomized algorithms.
  - $O(1.334^n)$, by Schöning's multi-restart random walk algorithm. [Schöning,99FOCS]
  - $O(1.323^n)$, by a combination of the above algorithm and the PPSZ algorithm [Rolo6].
- Deterministic algorithms.
  - $O(1.5^n)$, by the cube-covering-based algorithm [DantsinGoerdtHirschKannanKleinbergPapadimitriouRaghavanSchöning,02TCS].
  - $O(1.473^n)$ The above bound can be improved using a pruning technique for search inside a ball. The currently best bound based on this method is $O(1.473^n)$. [BrueggemannKern,04TCS]

Theorem ([PPSZ05]). *The PPSZ algorithm solves k-SAT in time*

$$|F|^{O(1)} \cdot 2^{n\left(1-\frac{\mu_k}{k-1}+o(1)\right)}$$

with error probability $o(1)$, *where* μ_k *is an increasing sequence with* $\mu_3 \approx 1.227$ *and* $\lim_{k\to\infty} \mu_k = \pi^2/6$.

Taken from "Handbook of Satisfiability"

PPSZ: The current best algorithm for unique k-SAT with k>3
[PaturiPudlakSaksZane,05JACM]

Phase transition of random k-SAT

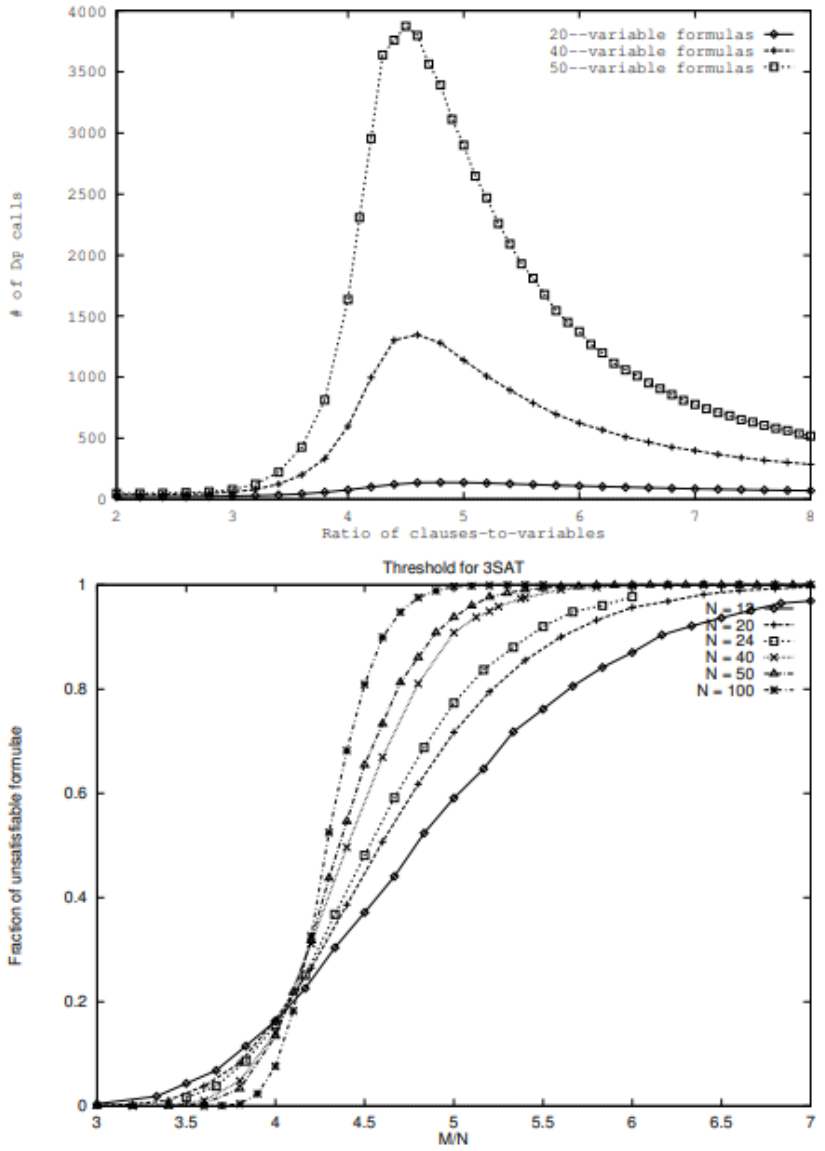
The phase transition phenomenon in random 3-SAT.

Top: the “easy-hard-easy” computational hardness w.r.t. DPLL, peaks at α ≈ 4.26.

Bottom: Formulas change from being mostly satisfiable to mostly unsatisfiable. The transitions sharpen as the number of variables grows.

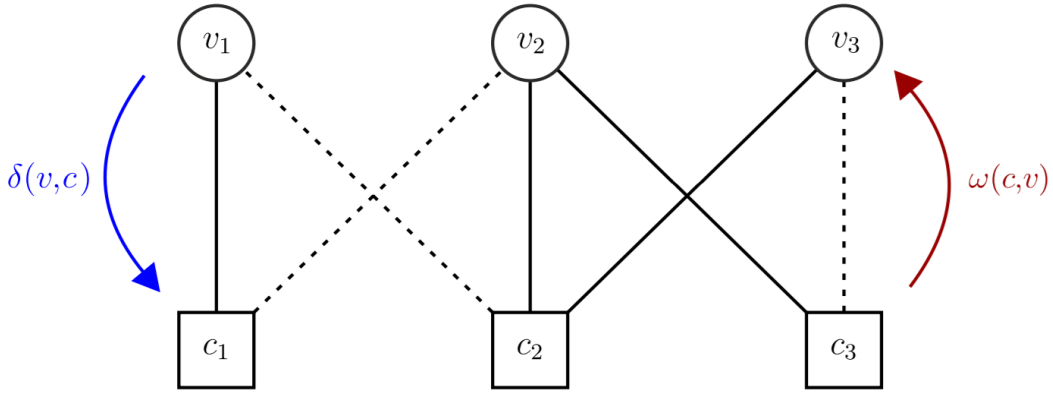
For general k, the threshold for random k-SAT is known to be in the range $2^k ln2 - O(k)$ [Achlioptas, Naor, Peres. 2005 Nature]

Solving hard phase transition random benchmarks becomes a major motivation on early the development of incomplete solvers, including stochastic local search (SLS) and survey propagation.



Phase transition and Survey Propagation (SP)

- Survey propagation
  - derived from the cavity method for spin glasses in statistical physics. [MézardParisiZecchina, 2002 Science]
- $F = (v_1 \vee \neg v_2) \wedge (\neg v_1 \vee v_2 \vee v_3) \wedge (v_2 \vee \neg v_3)$
- SP performs iterative calculations of messages



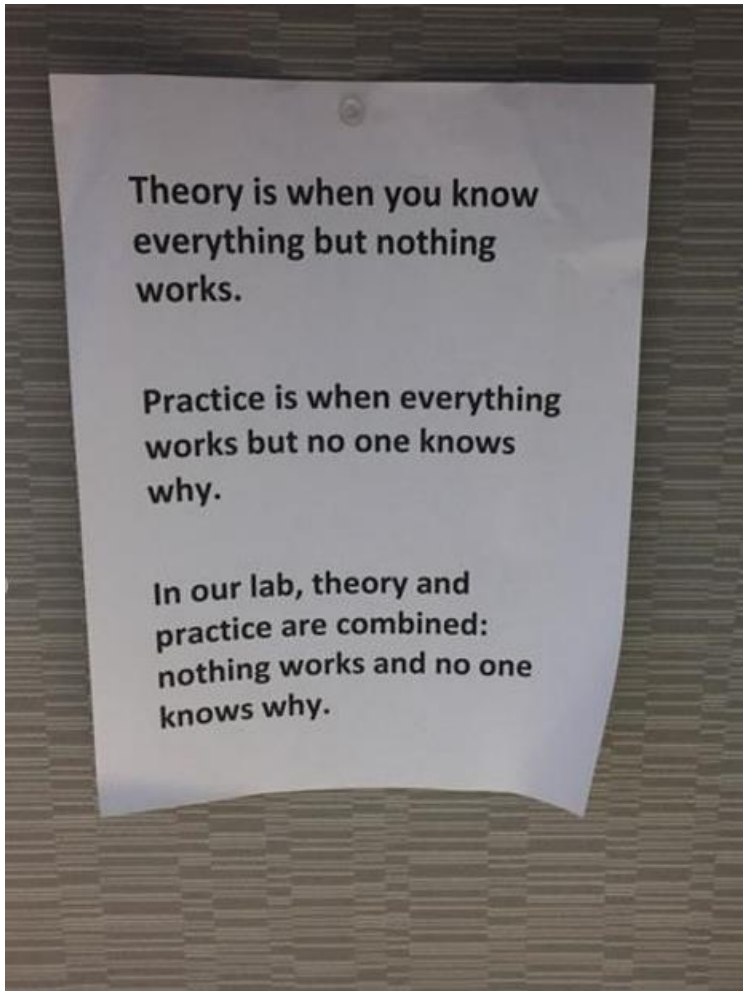
- Randomly and independently initialize $\delta(l, c) \in (0.0, 1.0)$, then repeat:
  - Update ω messages based on δ messages
  - Update δ messages based on ω messages

Phase transition and Survey Propagation (SP)

- The current best method for solving random satisfiable k-SAT close to phase transition is due to the combination SP+SLS.
  - SP is first called to simplify the original formula
  - SLS is then activated to solve the sub-formula after simplification
- a study of SP+SLS is presented at [LuoCaiWuSu,13CP]
- Dimetheus [Gableske, 2016SAT],

- SP+SLS solves huge random 3-SAT instances ($r = 4.2$, #var $= 10^7$) within 2 hours [LuoCaiWuSu,13CP]

Algorithm, Solver



Algorithms

vs.

Solvers

SAT revolution

SAT solvers,
programs that solve SAT formulas,
have become extremely powerful over the last two decades.

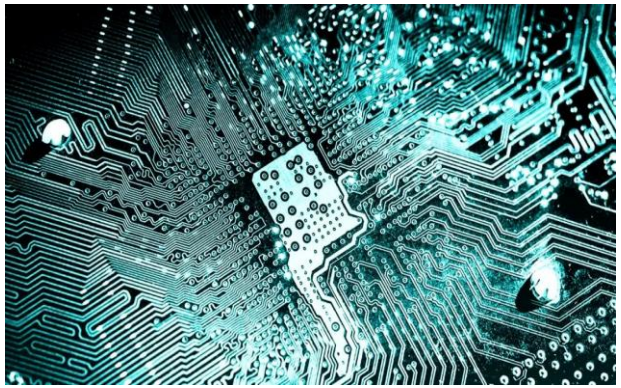
The Science of Brute Force

By Marijn J. H. Heule, Oliver Kullmann
Communications of the ACM, August 2017, Vol. 60 No. 8, Pages 70-79

Today's solvers can handle formulas with millions of variables, resulting in the *SAT revolution*
--- Solve problems in 3 steps:

SAT Encoding → *SAT Solving* → *Decoding*

SAT revolution



EDA

original C code

```
if(!a && !b) h();
else if(!a) g();
else f();

        ⇓

if(!a) {
  if(!b) h();
  else g();
} else f();
```

optimized C code

```
if(a) f();
else if(b) g();
else h();

        ⇑

if(a) f();
else {
  if(!b) h();
  else g(); }
```

How to check that these two versions are equivalent?

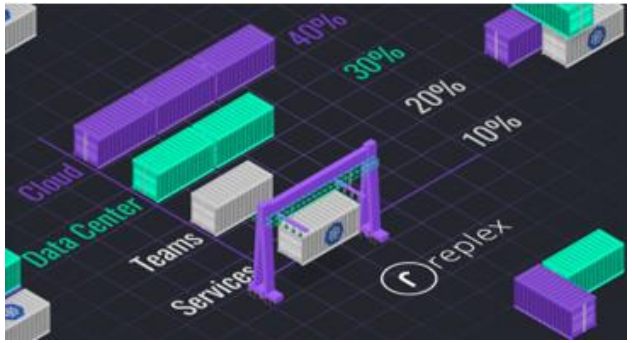
Program Analysis

Planning

cryptography

$x^m + y^m = z^m \pmod p$ *vdW(6) = 1132*
Schur's Theorem *Ramsey Theory*
Pythagorean Tuples Conjecture
3n+1 Conjecture?

Math



Resource Allocation

Using SAT Solvers

Input file: DIMACS format.

```
c example
p cnf 4 4
1 -4 -3 0
1 4 0
-1 0
-4 3 0
```

lines starting ”c” are comments and are ignored by the SAT solver
a line starting with ”p cnf” is the problem definition line containing the number of variables and clauses.
the rest of the lines represent clauses, literals are integers (starting with variable 1), clauses are terminated by a zero.

Output format

```
c comments, usually stastitics about the solving
s SATISFIABLE
v 1 2 -3 -4  5 -6 -7 8 9 10
v -11 12 13 -14 15 0
```

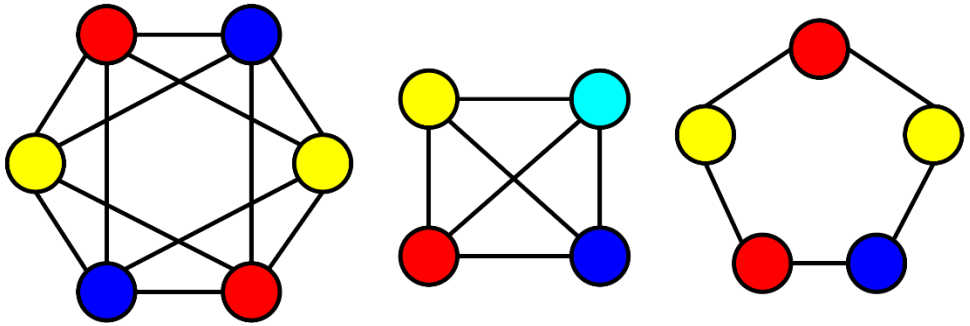
the solution line (starting with ”s”) can contain SATISFIABLE, UNSATISFIABLE and UNKNOWN.
For SATISFIABLE case, the truth values of variables are printed in lines starting with ”v”, the last value is followed by a ”0”

Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search
- Hybrid SAT Solving

SAT Encoding: graph coloring problem

A coloring is an assignment of colors to vertices such that no two adjacent vertices share the same color.



The Graph Coloring Problem (GCP) is to find a coloring of a graph while minimizing the number of colors.

The decision version: given a positive number k, decide whether a graph can be colored with k colors.

SAT Encoding: graph coloring problem

- 3-coloring problem:
  - for each vertex, uses *3* variables (n vertices), $4 \times 3 = 12$ in all.

    $x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23}, x_{31}, x_{32}, x_{33}, x_{41}, x_{42}, x_{43}$

  - *For each edge, produces 3 negative 2-clause*

    edge1-2: $\neg x_{11} \vee \neg x_{21}$, $\neg x_{12} \vee \neg x_{22}$, $\neg x_{13} \vee \neg x_{23}$;

    edge1-4: $\neg x_{11} \vee \neg x_{41}$, $\neg x_{12} \vee \neg x_{42}$, $\neg x_{13} \vee \neg x_{43}$;

    edge2-3: $\neg x_{21} \vee \neg x_{31}$, $\neg x_{22} \vee \neg x_{32}$, $\neg x_{23} \vee \neg x_{33}$;

    edge2-4: $\neg x_{21} \vee \neg x_{41}$, $\neg x_{22} \vee \neg x_{42}$, $\neg x_{23} \vee \neg x_{43}$;

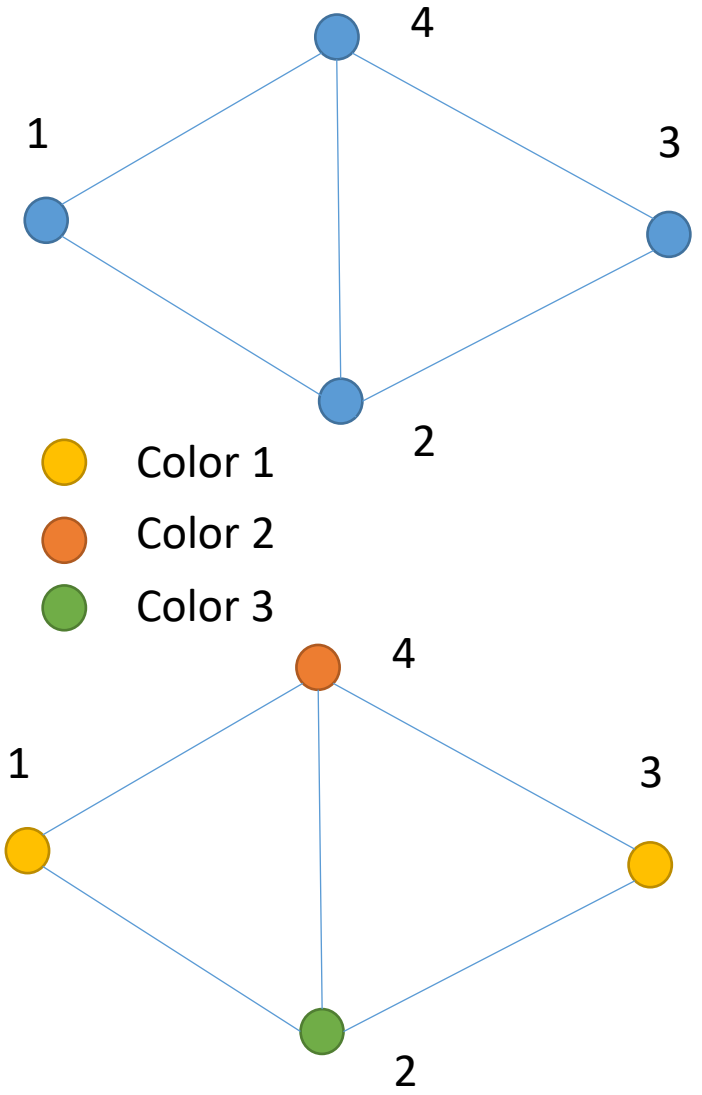
    edge3-4: $\neg x_{31} \vee \neg x_{41}$, $\neg x_{32} \vee \neg x_{42}$, $\neg x_{33} \vee \neg x_{43}$;

  - For each vertex, produces a positive *k*-clauses

    $x_{11} \vee x_{12} \vee x_{13}$, $x_{21} \vee x_{22} \vee x_{23}$, $x_{31} \vee x_{32} \vee x_{33}$, $x_{41} \vee x_{42} \vee x_{43}$

  - Result:

    $x_{11}, \neg x_{12}, \neg x_{13}, \neg x_{21}, x_{22}, \neg x_{23}, x_{31}, \neg x_{32}, \neg x_{33}, \neg x_{41}, \neg x_{42}, x_{43}$



SAT Encoding: logic puzzle

- Question: at least one of them speak truth. ho speaks the truth?
  - A: B is lying.
  - B: C is lying.
  - C: A and B is lying.

- Encoding:
  - 3 variables: a, b, c present A, B, C speak truth, while ¬a, ¬b, ¬c present lying.
  - clauses:

a ∨ b ∨ c ; %at least one speak truth.
¬ a ∨ ¬ b; a ∨ b; %a-> ¬b, ¬a -> b
¬ b ∨ ¬ c; b ∨ c; %b-> ¬c, ¬b -> c
¬ c ∨ ¬ a; ¬c ∨ ¬b; c ∨ a ∨ b %c->(¬a∧ ¬b), ¬c->¬ (¬a∧ ¬b)

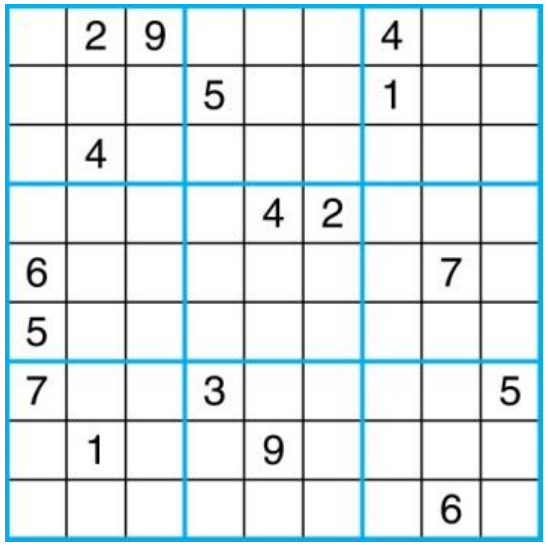
- Result: ¬a, b, ¬c

  B speaks truth, A and C are lying

Encoding Sudoku to SAT

- A **Sudoku puzzle** is represented by a 9×9 grid made up of nine 3×3 blocks. Some of the 81 cells of the puzzle are assigned one of the numbers 1,2, …, 9.
- Goal: assign numbers to each blank cell so that every row, column and block contains each of the nine possible numbers.

- Let $p(i,j,n)$ denote the proposition that is true when the cell in the i-th row and the j-th column has number n.
- There are 9×9 × 9 = 729 such propositions.
- In the sample puzzle p(5,1,6) is true, but p(5,j,6) is false for j = 2,3,…9



Encoding Sudoku to SAT

- For each cell with a given value, assert $p(i,j,n)$, when the cell in row i and column j has the given value.

- Assert that every row contains every number.

$$\bigwedge_{i=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{j=1}^{9} p(i,j,n)$$

- Assert that every column contains every number.

$$\bigwedge_{j=1}^{9} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{9} p(i,j,n)$$

	2	9				4		
			5			1		
	4							
				4	2			
6							7	
5								
7			3					5
	1			9				
							6	

Encoding Sudoku to SAT

- Assert that each of the 3 x 3 blocks contain every number.

$$\bigwedge_{r=0}^{2} \bigwedge_{s=0}^{2} \bigwedge_{n=1}^{9} \bigvee_{i=1}^{3} \bigvee_{j=1}^{3} p(3r+i, 3s+j, n)$$

- Assert that no cell contains more than one number.

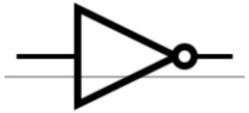
$$n \neq n'$$

$$p(i,j,n) \rightarrow \neg p(i,j,n')$$

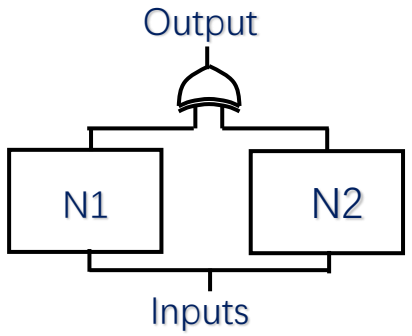
	2	9				4		
			5			1		
	4							
				4	2			
6							7	
5								
7			3					5
	1			9				
							6	

Circuit to CNF

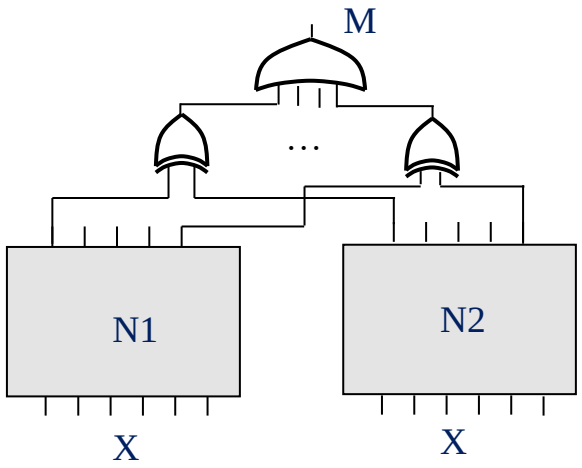
Tseitin Transformation

Type	Operation	CNF Sub-expression
 AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
 NAND	$C = \overline{A \cdot B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
 OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
 NOR	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
 NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
 XOR	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$

SAT Encoding: Equivalence Checking



Single output



Multi-output

- Build a miter circuit
- Transform Miter Circuit to CNF
- Call a SAT solver

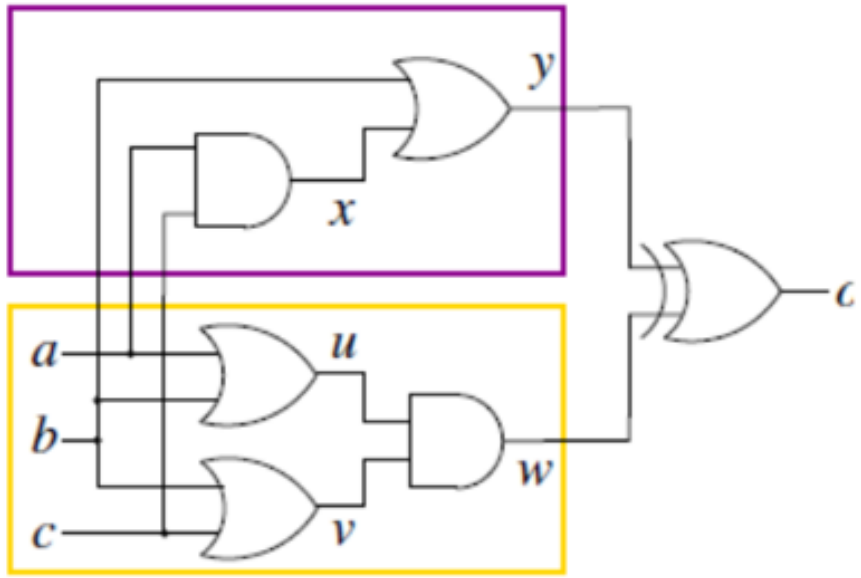
  If SAT，we find a counter-example

  If UNSAT，N1=N2

N1 xor N2 → CNF

SAT Encoding: Equivalence Checking

A small example of EC



$$
\begin{aligned}
& o \wedge \\
& (x \leftrightarrow a \wedge c) \wedge \\
& (y \leftrightarrow b \vee x) \wedge \\
& (u \leftrightarrow a \vee b) \wedge \\
& (v \leftrightarrow b \vee c) \wedge \\
& (w \leftrightarrow u \wedge v) \wedge \\
& (o \leftrightarrow y \oplus w)
\end{aligned}
$$

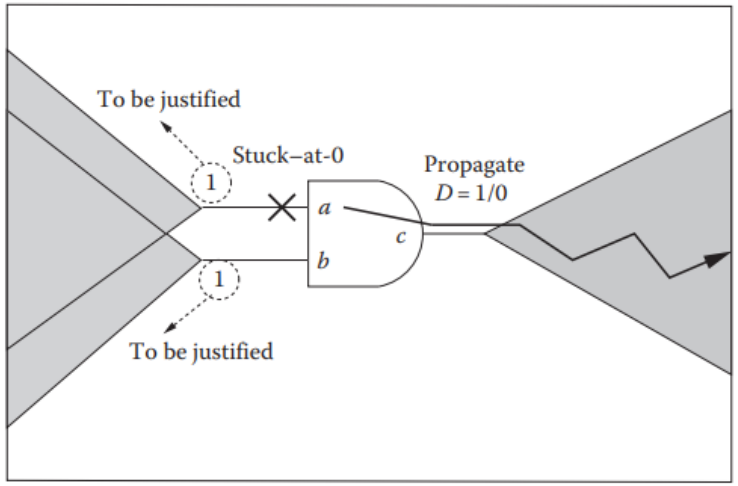
$$o \wedge (x \rightarrow a) \wedge (x \rightarrow c) \wedge (x \leftarrow a \wedge c) \wedge \ldots$$

$$o \wedge (\overline{x} \vee a) \wedge (\overline{x} \vee c) \wedge (x \vee \overline{a} \vee \overline{c}) \wedge \ldots$$

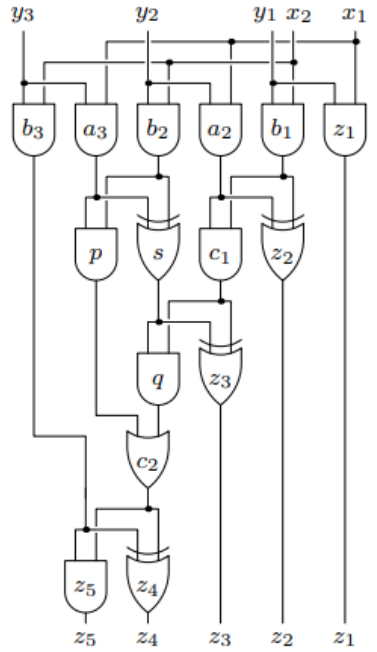
SAT Encoding: ATPG

ATPG （Automatic Test Pattern Generation）

- After producing a chip the functional correctness of the integrated circuit has to be checked.
  - Consider the stuck-at *fault model*



$$
\begin{array}{r}
y_3\ y_2\ y_1 \\
\times \quad x_2\ x_1 \\
\hline
a_3\ a_2\ a_1 \\
b_3\ b_2\ b_1 \\
\hline
c_3\ c_2\ c_1 \\
\hline
z_5\ z_4\ z_3\ z_2\ z_1
\end{array}
$$

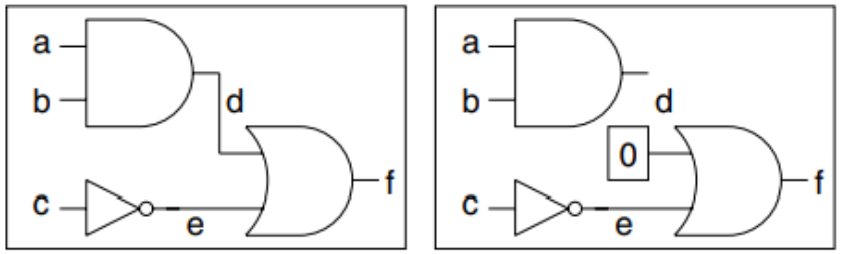


50 wires，
100 possible stuck-at faults

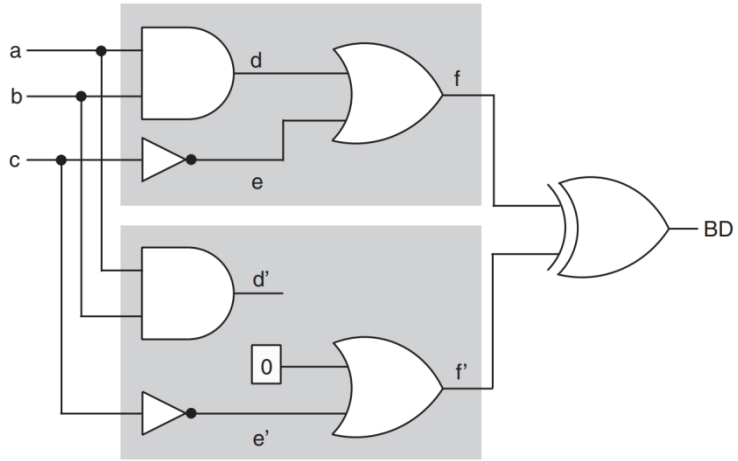
SAT Encoding: ATPG

ATPG （Automatic Test Pattern Generation）

- After producing a chip the functional correctness of the integrated circuit has to be checked.

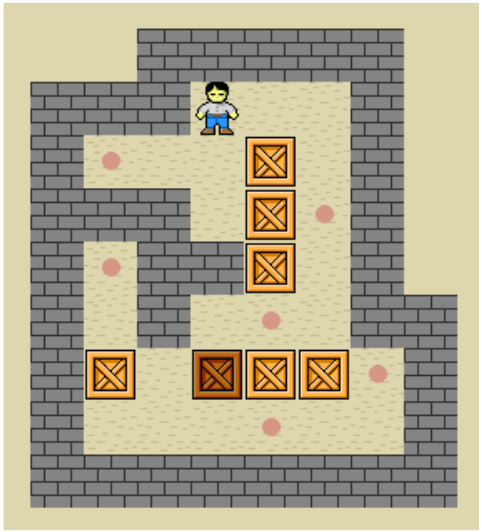


(a) Correct circuit (b) Faulty circuit



Encoding Planning to SAT

- Variables – For each location we have variable, the domain is WORKER, BOX, EMPTY
- Initial State – assign values based on the picture
- Goal – goal position variables have value BOX
- Actions – move and push for each possible location
  - push(L1; L2; L3)
    = ({L1 = W; L2 = B; L3 = E}; {L1 = E; L2 = W; L3 = B}).
  - move(L1; L2) = ({L1 = W; L2 = E}; {L1 = E; L2 = W})



- We cannot encode the existence of a plan in general
- But we can encode the existence of plan up to some length

[example taken from SAT lecture by Carsten Sinz, Toma´s Balyo]

Encoding Planning to SAT

- A planning problem instance Π is a tuple (χ, A, S_I, S_G) where
- χ is a set of multivalued variables with finite domains.

  each variable $x \in \chi$ has a finite possible set of values $dom(x)$
- A is a set actions. Each action $a \in A$ is a tuple $\big(pre(a), eff(a)\big)$

  $pre(a)$ is a set of preconditions of action a

  $eff(a)$ is a set of effects of action a

  both are sets of equalities of the form $x = v$ where $x \in \chi$ and $v \in dom(x)$
- s_I is the initial state, it is a **full** assignment of the variables in χ
- s_G is the set of goal conditions, it is a set of equalities(same as $pre(a)$ and $eff(a)$)

Encoding Planning to SAT

The task

- Given a planning problem instance $\Pi = (\chi, A, S_I, S_G)$ and $k \in \mathrm{N}$ construct a CNF formula F sus that F satisfiable if and only if there is plan of length k for Π.

We need two kinds of variables

- Variables to encode the actions:

  a_i^t for each $t \in \{1, \dots, k\}$ and $a_i \in A$

- Variables to encode the states:

  $b_{x=v}^t$ for each $t \in \{1, \dots, k+1\},\ x \in \chi\ and\ v \in dom(x)$

- In total we have $k|A| + (k+1)\sum_{x \in \chi} dom(x)$ variables

Encoding Planning to SAT

We will need 8 kinds of clauses

- The first state is the initial state
- The goal conditions are satisfied in the end

- Each state variable has at least one value
- Each state variable has at most one value

- If an action is applied it must be applicable
- If an action is applied its effects are applied in the next step
- State variables cannot change without an action between steps

- At most one action is used in each step

Encoding Planning to SAT

The first state is the initial state:

$$(b^1_{x=v})$$

$$\forall (x = v) \in s_I$$

The goal conditions are satisfied in the end:

$$(b^{k+1}_{x=v})$$

$$\forall (x = v) \in s_G$$

Encoding Planning to SAT

Each state variable has at least one value:

$$(b^t_{x=v_1} \vee b^t_{x=v_2} \vee \cdots b^t_{x=v_d})$$

$$\forall x \in \chi, dom(x) = \{v_1, v_1, \ldots, v_d\}, \forall t \in \{1, \ldots, k+1\}$$

Each state variable has at most one value:

$$(\neg b^t_{x=v_i} \vee \neg b^t_{x=v_j})$$

$$\forall x \in \chi, v_i \neq v_j, \{v_i, v_j\} \subseteq dom(x), \forall t \in \{1, \ldots, k+1\}$$

Encoding Planning to SAT

If an action is applied it must be applicable:

$$(\neg a^t \vee b^t_{x=v})$$

$$\forall a \in A, \forall (x = v) \in pre(a), \forall t \in \{1, \ldots, k\}$$

If an action is applied its effects are applied in the next step:

$$(\neg a^t \vee b^{t+1}_{x=v})$$

$$\forall a \in A, \forall (x = v) \in eff(a), \forall t \in \{1, \ldots, k\}$$

Encoding Planning to SAT

State variables cannot change without an action between steps.

If $x \neq v$ at t, but $x = v$ at t+1, then some action supporting $x = v$ must happen.

$$(\neg b_{x=v}^{t+1} \vee b_{x=v}^{t} \vee a_{s_1}^{t} \vee \cdots \vee a_{s_j}^{t})$$

$$\forall x \in \chi, \forall v \in dom(x), support(x = v) = \{a_{s_1}, \ldots, a_{s_j}\}, \forall t \in \{1, \ldots, k\}$$

By support$(x = v) \subseteq A$ we mean the set of supporting actions of the assignment $x = v$,i.e., the set of actions that have $x = v$ as one of their effects.

Encoding Planning to SAT

At most one action is used in each step:

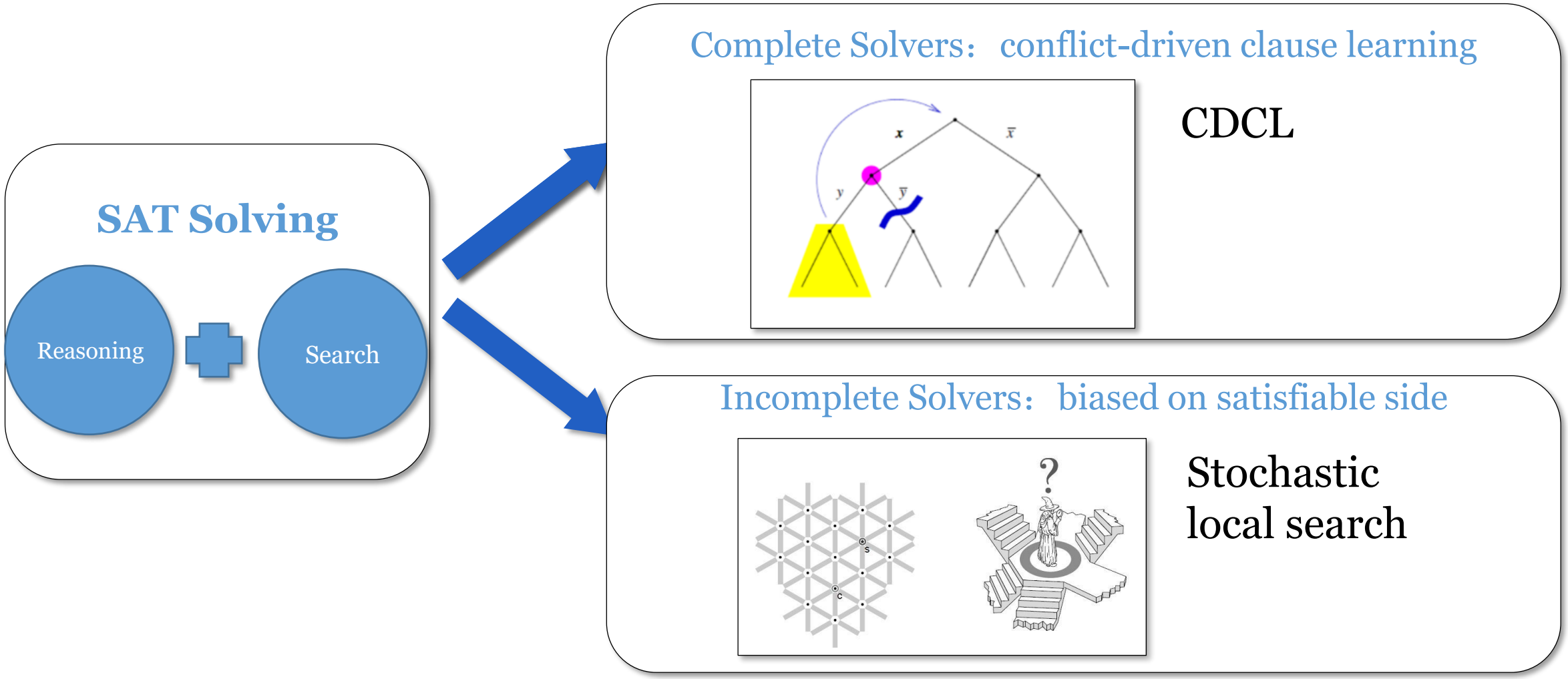
$$(\neg a_i^t \vee \neg a_j^t)$$

$$\forall \{a_i, a_j\} \subseteq A, a_i \neq a_j \; \forall t \in \{1, \ldots, k\}$$

Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search
- Hybrid SAT Solving

SAT Solving Basis



SAT Solving Basis - Resolution

Algorithm 1: Saturation Algorithm

```
Input: CNF formula F
Output: SAT,UNSAT
1  while true do
2      R = resolveAll(F)
3      if R ∩ F ≠ R then
4          F = F ∪ R
5      else
6          break
7  if ⊥∈ F then
8      return UNSAT
9  else
10     return SAT
```

$$\frac{C \vee x \qquad \bar{x} \vee D}{C \vee D}\ [\text{Res}]$$

- **Resolution.** If two clauses A and B have exactly one pair of complementary literals $a \in A$ and $\neg a \in B$, then the clause $A \cup B \backslash \{a, \neg a\}$ is called the resolvent of A and B (by a) and denoted by $R(A, B)$.
- This algorithm is sound and complete – always terminates and answers correctly
- Has exponential time and space complexity (always for Pigeons)

SAT Solving Basis - Resolution

Variable elimination by resolution

- Given a formula F and a literal a, the formula denoted $DP_a(F)$ is constructed from F
  - by adding all resolvents by a
  - and then removing all clauses that contain a or $\neg a$

Example. $F = (x \vee e) \wedge (y \vee e) \wedge (\bar{x} \vee z \vee \bar{e}) \wedge (y \vee \bar{e}) \wedge (y \vee z)$

Eliminating variable e by resolution:

- first add all resolvents upon e.

$\{(x \vee e), (y \vee e)\}$ with $\{(x \vee z \vee \bar{e}), (y \vee \bar{e})\}$ → 4 resolvents

$$F \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y) \wedge (y \vee \bar{x} \vee z) \wedge (y)$$

- remove all clauses that contain e to obtain

$$(y \vee z) \wedge (x \vee \bar{x} \vee z) \wedge (x \vee y)\,(y \vee \bar{x} \vee z) \wedge (y)$$

SAT Solving Basis - Unit Propagation

- Unit Clause: A Clause that all literals are falsified except one unassigned literal.
- Unit Propagation (UP): the unassigned literal in unit clause can only be assigned to single value to satisfy the clause.

Example:

$Vars$: x_1 T, x_2, x_3, x_4, x_5, x_6

$clauses$: sat

x_1

$\bar{x}_1 \vee \bar{x}_2$

$\bar{x}_1 \vee \bar{x}_3$

$x_2 \vee \bar{x}_3 \vee \bar{x}_4$

$\bar{x}_1 \vee x_4 \vee x_5$

$x_2 \vee x_4 \vee x_6$

SAT Solving Basis - Unit Propagation

- Unit Clause: A Clause that all literals are falsified except one unassigned literal.
- Unit Propagation (UP): the unassigned literal in unit clause can only be assigned to single value to satisfy the clause.

Example:

	x_1	T		x_1 sat
	x_2			$\bar{x}_1 \vee \bar{x}_2$
$Vars$:	x_3		$clauses$:	$\bar{x}_1 \vee \bar{x}_3$
	x_4			$x_2 \vee \bar{x}_3 \vee \bar{x}_4$
	x_5			$\bar{x}_1 \vee x_4 \vee x_5$
	x_6			$x_2 \vee x_4 \vee x_6$

SAT Solving Basis – DP Algorithm

Question: Can we do better than saturation-based resolution?

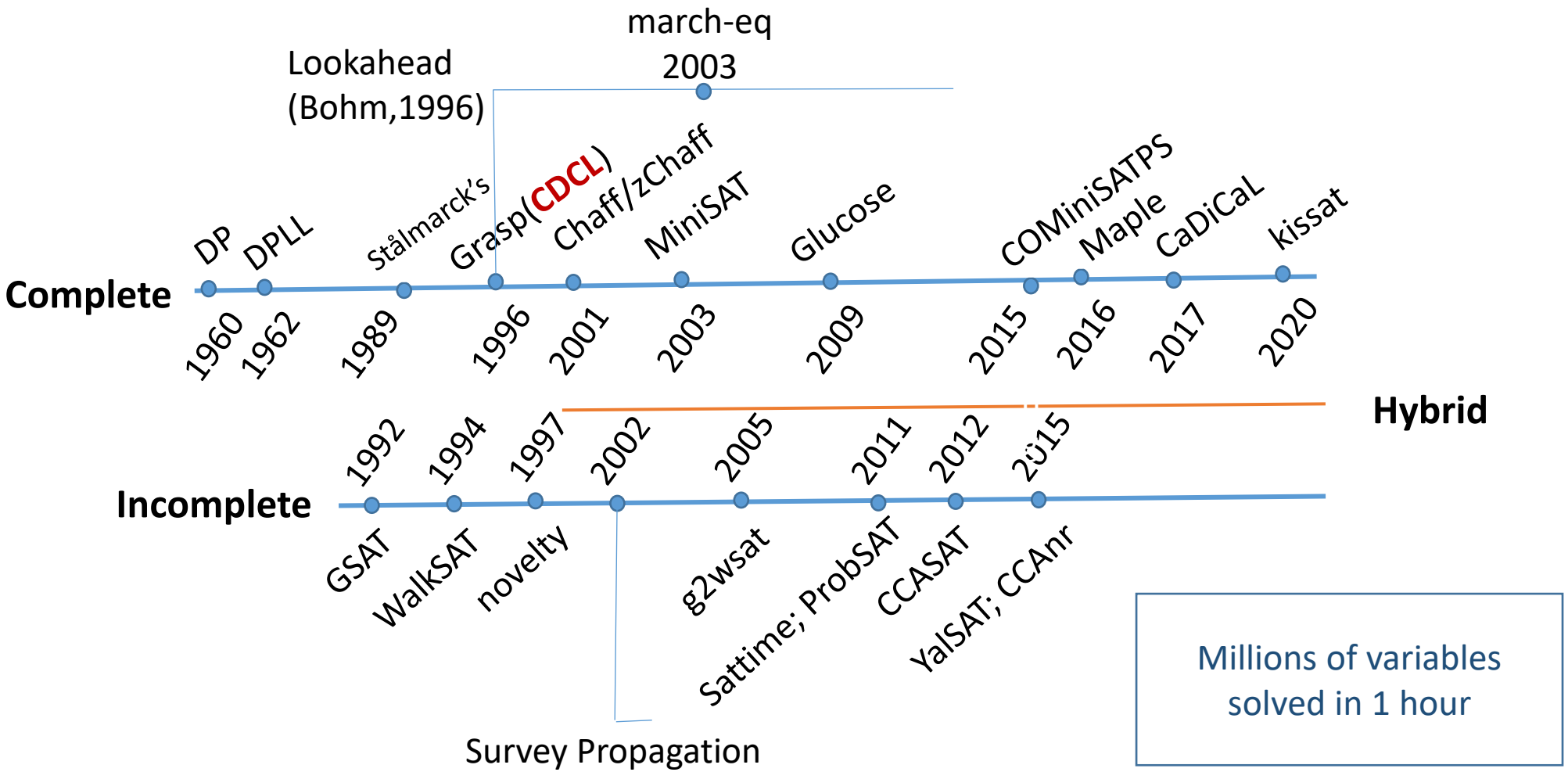
➔**Davis-Putnam Algorithm [1960]** % could find record in 1959

- Rule 1: Unit propagation
- Rule 2: Pure literal elimination
- Rule 3: Resolution at one variable

Apply deduction rules (giving priority to rules 1 and 2) until no further rule is applicable

Solver = Algorithmic framework + heuristics. [just a quick thinking, don't quote me…]

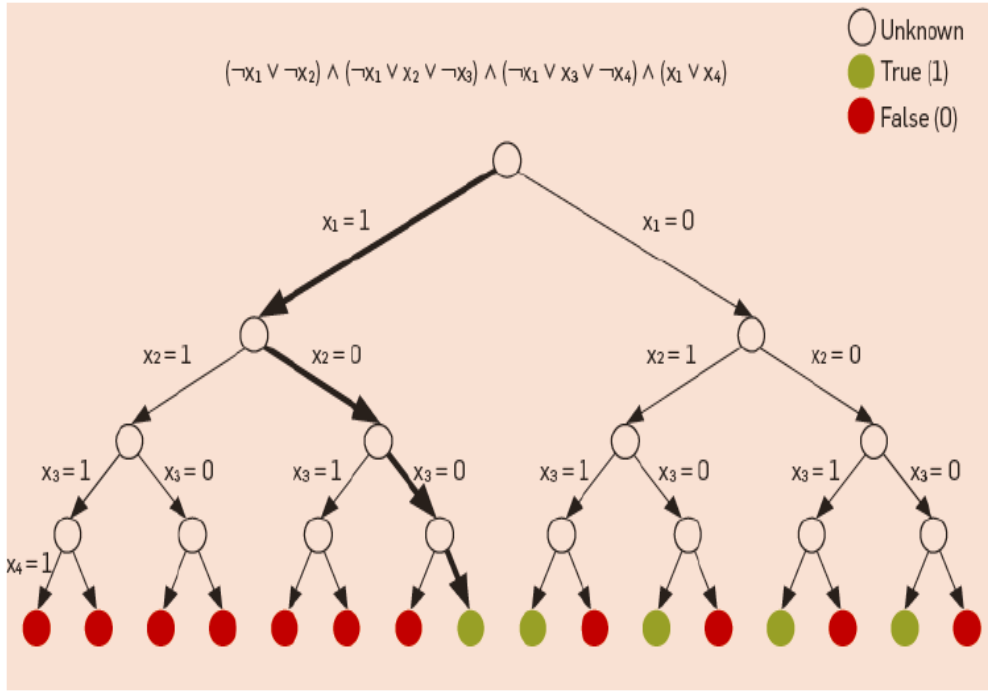
SAT Solving – Quest for Efficient SAT solving



DPLL Algorithm

Davis-Putnam-Logemann-Loveland (DPLL, 1962)

- Chronological backtracking + UP + Decision heuristics



Algorithm 3.4 DPLL(CNF Δ)

```
1:  (I, Γ) =UNIT-RESOLUTION(Δ)
2:  if Γ = {} then
3:     return I
4:  else if {} ∈ Γ then
5:     return UNSATISFIABLE
6:  else
7:     choose a literal L in Γ
8:     if L =DPLL(Γ|L) ≠ UNSATISFIABLE then
9:        return L ∪ I ∪ {L}
10:    else if L =DPLL(Γ|¬L) ≠UNSATISFIABLE then
11:       return L ∪ I ∪ {¬L}
12:    else
13:       return UNSATISFIABLE
```

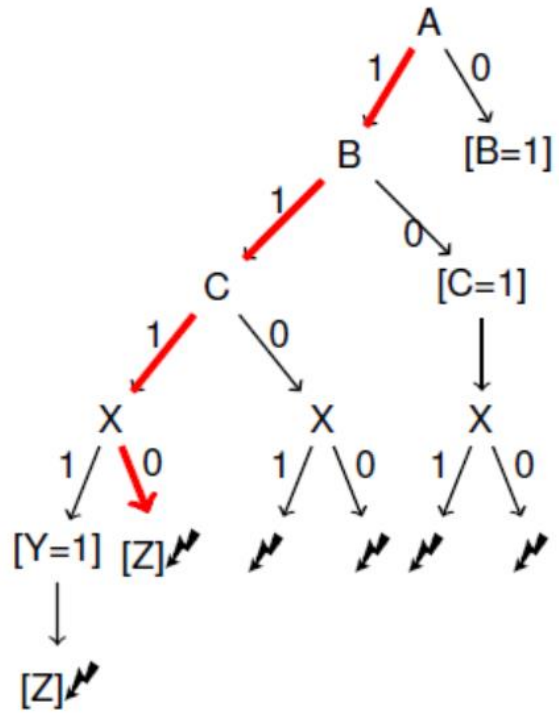
conditioning Δ on literal L: $\Delta|L = \{\alpha - \{\neg L\} \mid \alpha \in \Delta,\ L \notin \alpha\}.$

DPLL Algorithm

Davis-Putnam-Logemann-Loveland (DPLL, 1962)

- Chronological backtracking + UP + Decision heuristics

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$



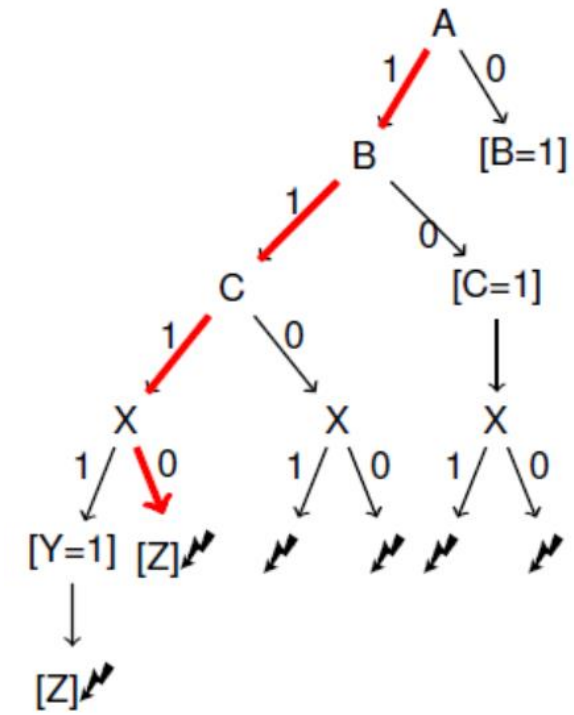
Decision level

[UP] Decide [UP] Decide [UP] ….

Level 0 Level 1

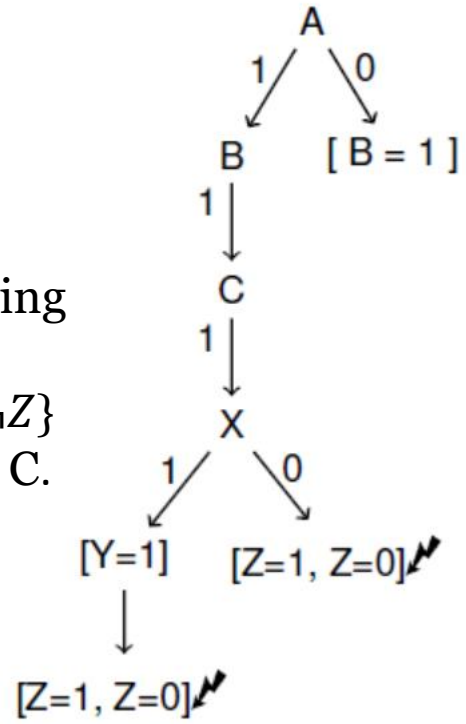
Backjumping

$$\Delta = \begin{array}{l} 1.\ \{A, B\} \\ 2.\ \{B, C\} \\ 3.\ \{\neg A, \neg X, Y\} \\ 4.\ \{\neg A, X, Z\} \\ 5.\ \{\neg A, \neg Y, Z\} \\ 6.\ \{\neg A, X, \neg Z\} \\ 7.\ \{\neg A, \neg Y, \neg Z\} \end{array}$$



Chronological Backtracking

The first two conflicting clauses $\{\neg A, \neg X, Y\}, \{\neg A, X, \neg Z\}$ do not involve B and C.



Non-Chronological Backtracking

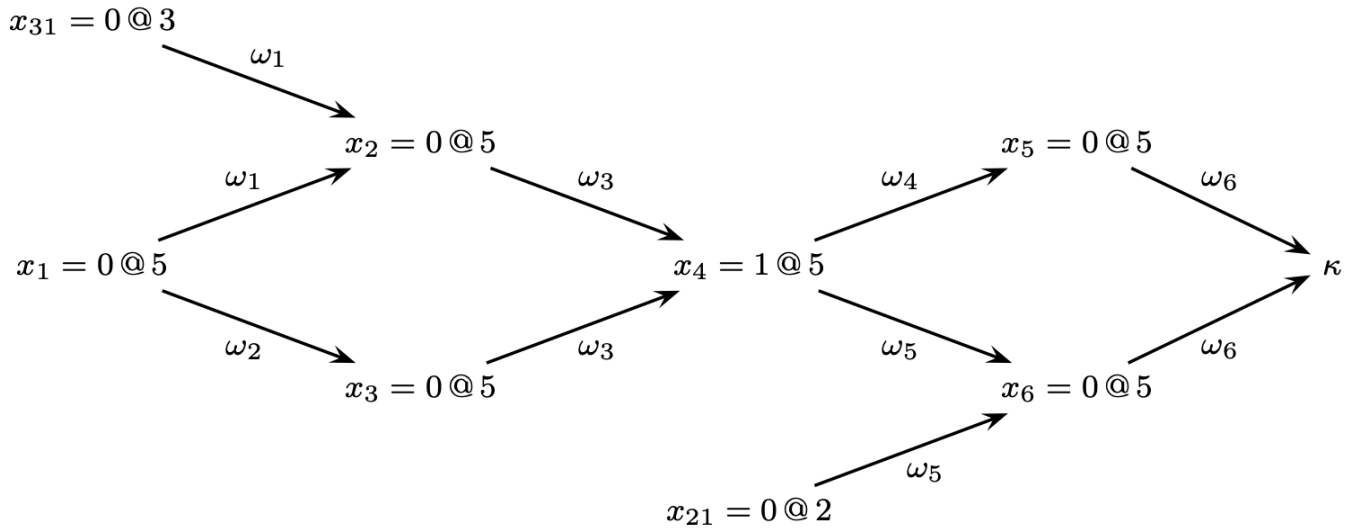
CDCL – Implication Graph

[MarqueSilvaSakallah' 96]

Implication Graph describes the decision and reasoning path.

- Vertex: (decision variable = value @decision level)
- Edge : unit clause used in UP (Reason Clause) .
- Conflict: all literals are falsified (under the current assignment).

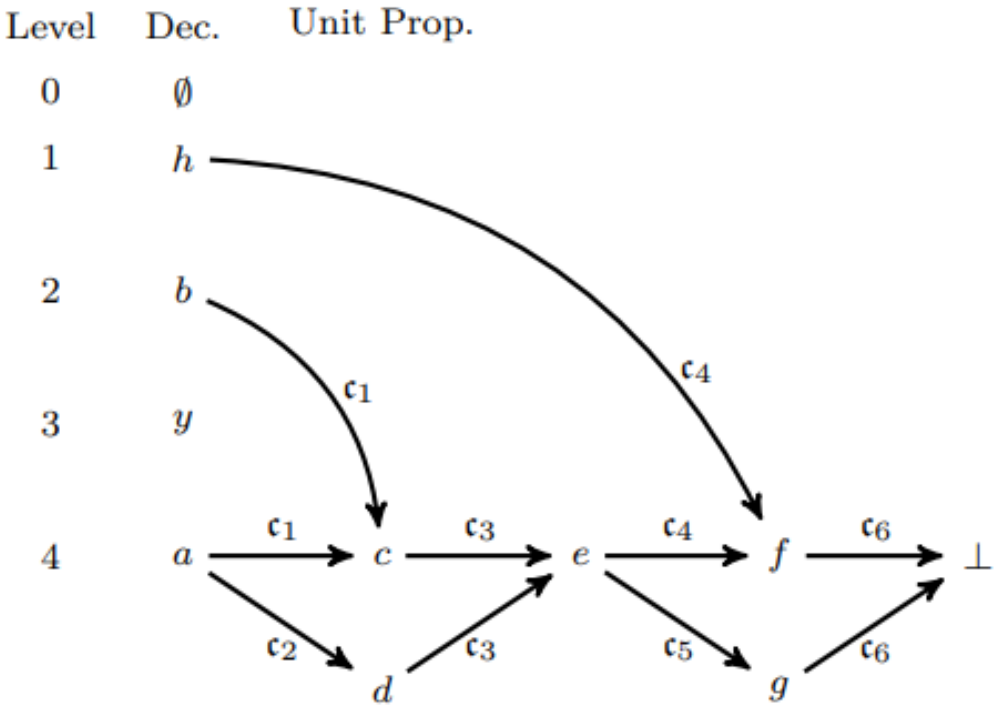
$$\begin{aligned}\varphi_1 &= \omega_1 \wedge \omega_2 \wedge \omega_3 \wedge \omega_4 \wedge \omega_5 \wedge \omega_6 \\ &= (x_1 \vee x_{31} \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge \\ &\quad (\neg x_4 \vee \neg x_5) \wedge (x_{21} \vee \neg x_4 \vee \neg x_6) \wedge (x_5 \vee x_6)\end{aligned}$$



CDCL – Implication Graph

Implication Graph

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



(a) Implication graph

$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

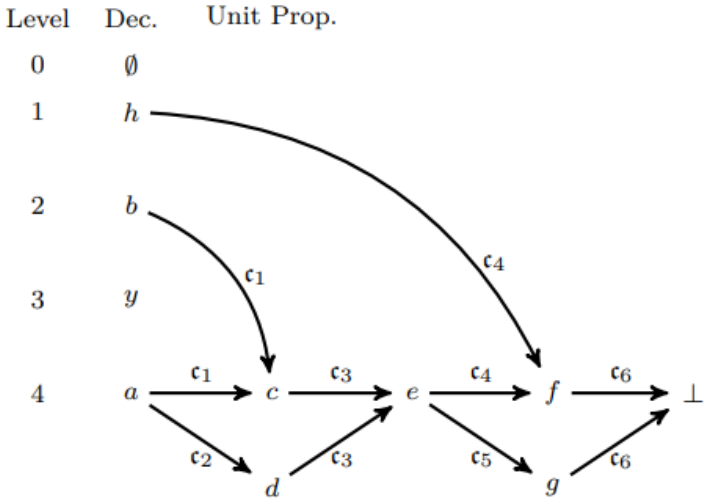
(b) State of variables

For variable x:

$v(x)$: *the value*
$\delta(x)$: *the decision level*
$\alpha(x)$: *the reason clause*

CDCL – Conflict Analysis

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



(a) Implication graph

$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

(b) State of variables

First UIP clause learning

Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	$\mathfrak{c}_4$	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	$\mathfrak{c}_5$	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	$\mathfrak{c}_3$	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	$[]$	–	–	$\{\bar{h}, \bar{e}\}$	–

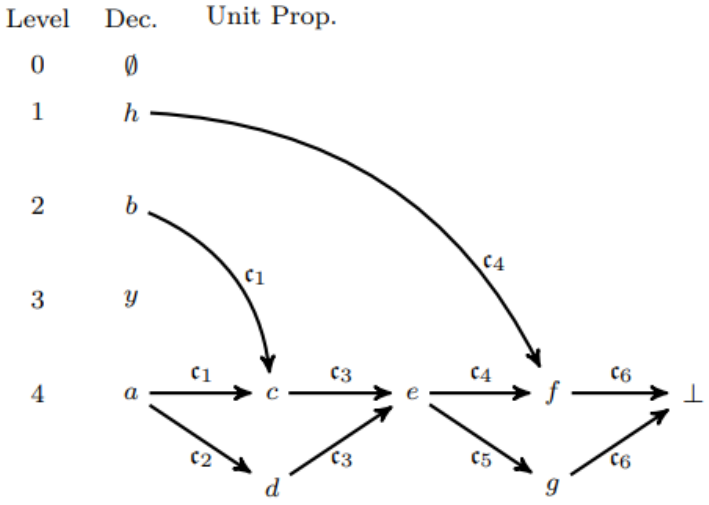
Variables are analyzed in a first-in first-out fashion, starting from the conflict.

unique implication point (UIP)
in step 3, there exists only one variable to trace, e, it is a UIP.
a UIP is a dominator of the decision variable with respect to the conflict node $\bot$.

Any literals assigned at decision levels smaller than the current one are added to (i.e. recorded in) the clause being learned

CDCL – Conflict Analysis

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



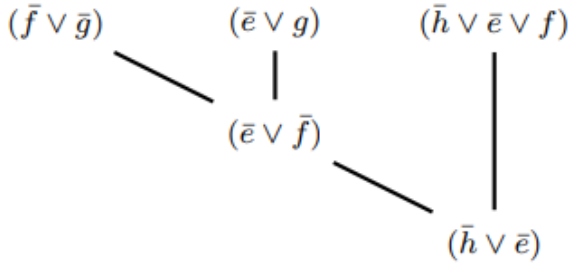
(a) Implication graph

$x \in \mathbb{V} \cup \{\bot\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
h	1	1	$\mathfrak{d}$
b	1	2	$\mathfrak{d}$
y	1	3	$\mathfrak{d}$
a	1	4	$\mathfrak{d}$
c	1	4	$\mathfrak{c}_1$
d	1	4	$\mathfrak{c}_2$
e	1	4	$\mathfrak{c}_3$
f	1	4	$\mathfrak{c}_4$
g	1	4	$\mathfrak{c}_5$
$\bot$	–	–	$\mathfrak{c}_6$

(b) State of variables

First UIP clause learning

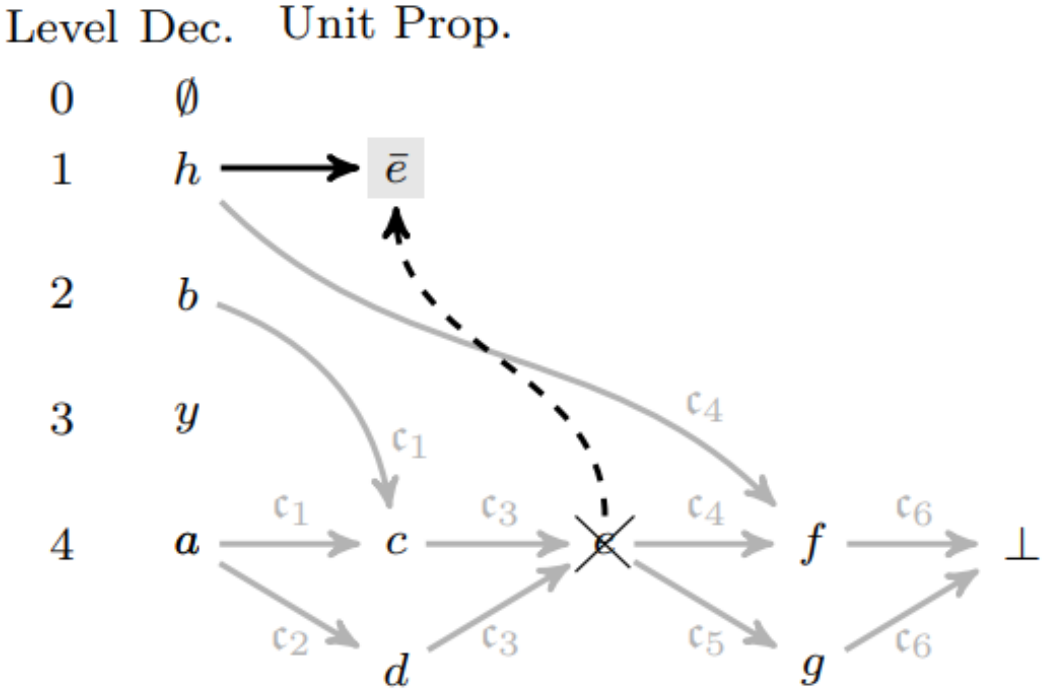
Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\bot$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
1	$[f, g]$	f	$\mathfrak{c}_4$	$\{\bar{h}\}$	$\{e\}$
2	$[g, e]$	g	$\mathfrak{c}_5$	$\{\bar{h}\}$	$\emptyset$
3	$[e]$	e	$\mathfrak{c}_3$	$\{\bar{h}, \bar{e}\}$	$\emptyset$
6	[]	–	–	$\{\bar{h}, \bar{e}\}$	–



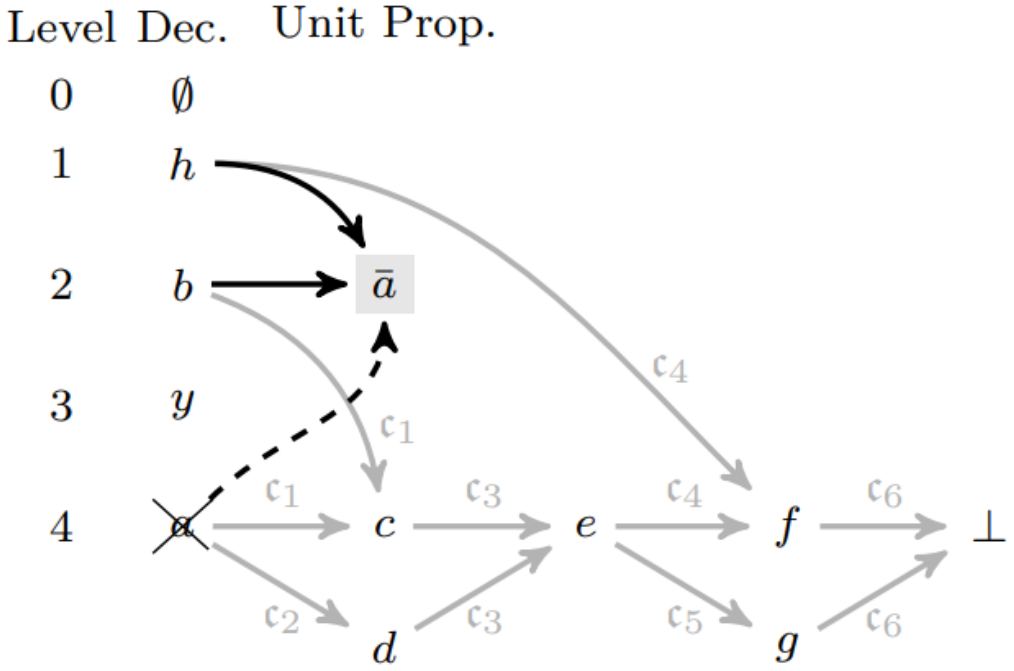
Resolution steps with first UIP learning

CDCL – Non-Chronological backtracking with conflict analysis

$$\mathcal{F}_2 = \mathfrak{c}_1 \wedge \mathfrak{c}_2 \wedge \mathfrak{c}_3 \wedge \mathfrak{c}_4 \wedge \mathfrak{c}_5 \wedge \mathfrak{c}_6$$
$$= (\bar{a} \vee \bar{b} \vee c) \wedge (\bar{a} \vee d) \wedge (\bar{c} \vee \bar{d} \vee e) \wedge (\bar{h} \vee \bar{e} \vee f) \wedge (\bar{e} \vee g) \wedge (\bar{f} \vee \bar{g})$$



NBC with first UIP learnt clause $\overline{h} \vee \overline{e}$

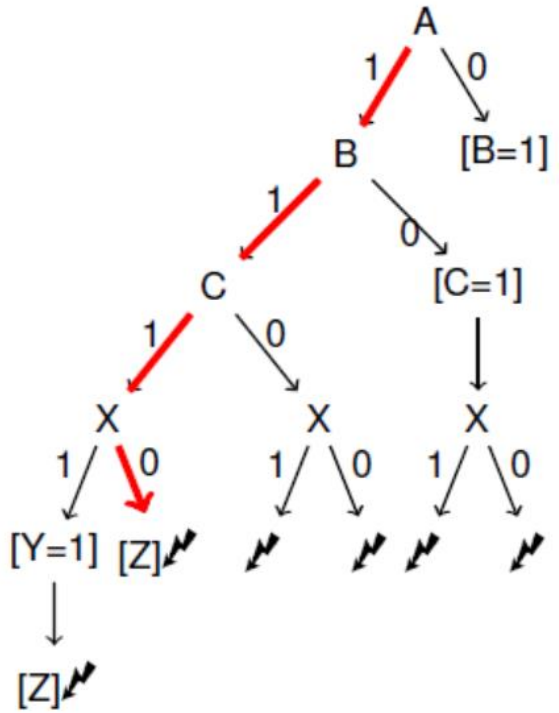


NBC with learnt clause $\overline{h} \vee \overline{b} \vee \overline{a}$

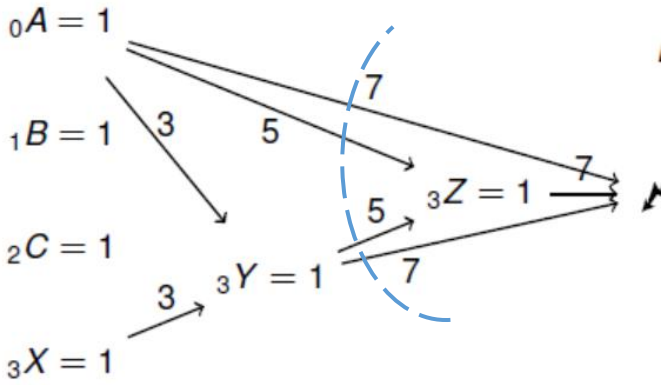
CDCL – Non-Chronological backtracking with conflict analysis

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$

$\Delta =$
1. $\{A, B\}$
2. $\{B, C\}$
3. $\{\neg A, \neg X, Y\}$
4. $\{\neg A, X, Z\}$
5. $\{\neg A, \neg Y, Z\}$
6. $\{\neg A, X, \neg Z\}$
7. $\{\neg A, \neg Y, \neg Z\}$
8. $\{\neg A, \neg Y\}$



Chronological Backtracking

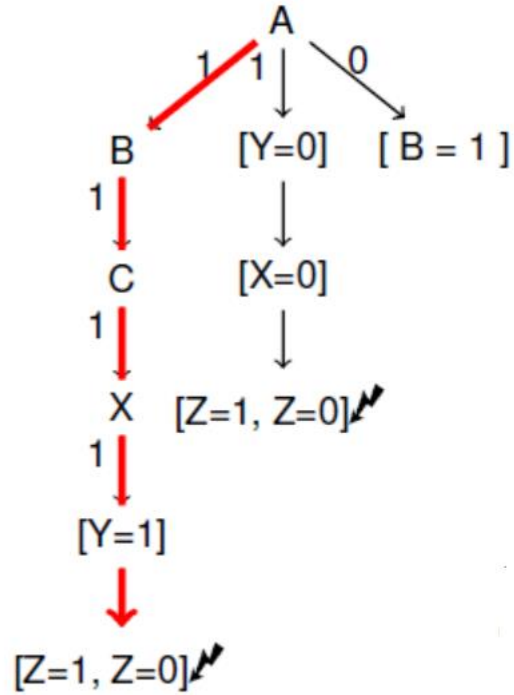


Conflict Analysis

Conflicting Clause:$\{\neg A, \neg Y, \neg Z\}$

Learnt Clause(1UIP):$\{\neg A, \neg Y\}$

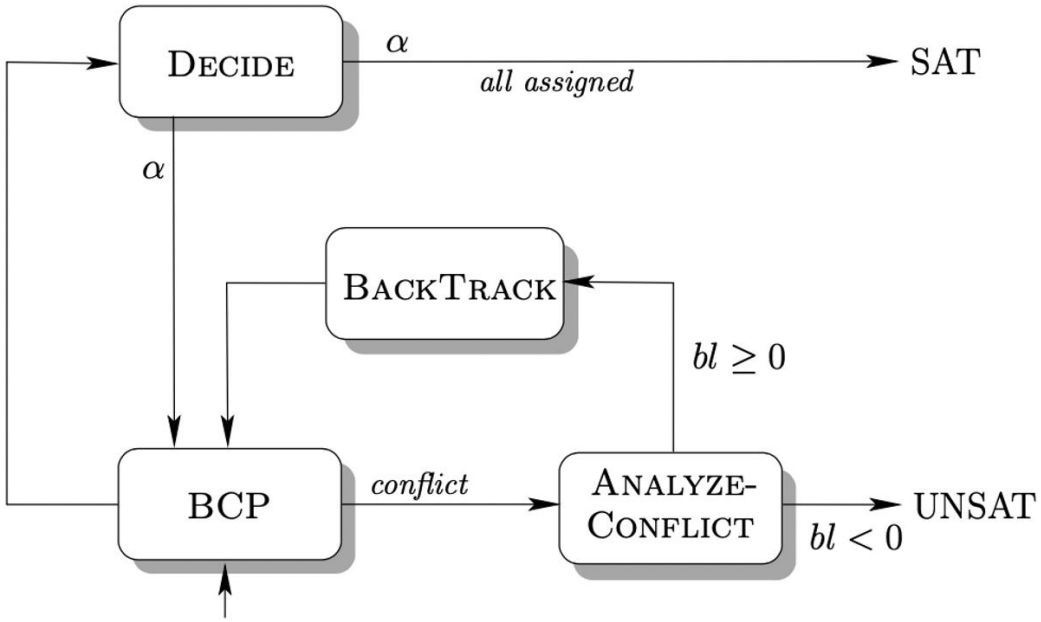
Clause Learning



Non-Chronological Backtracking

CDCL – Algorithm

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy

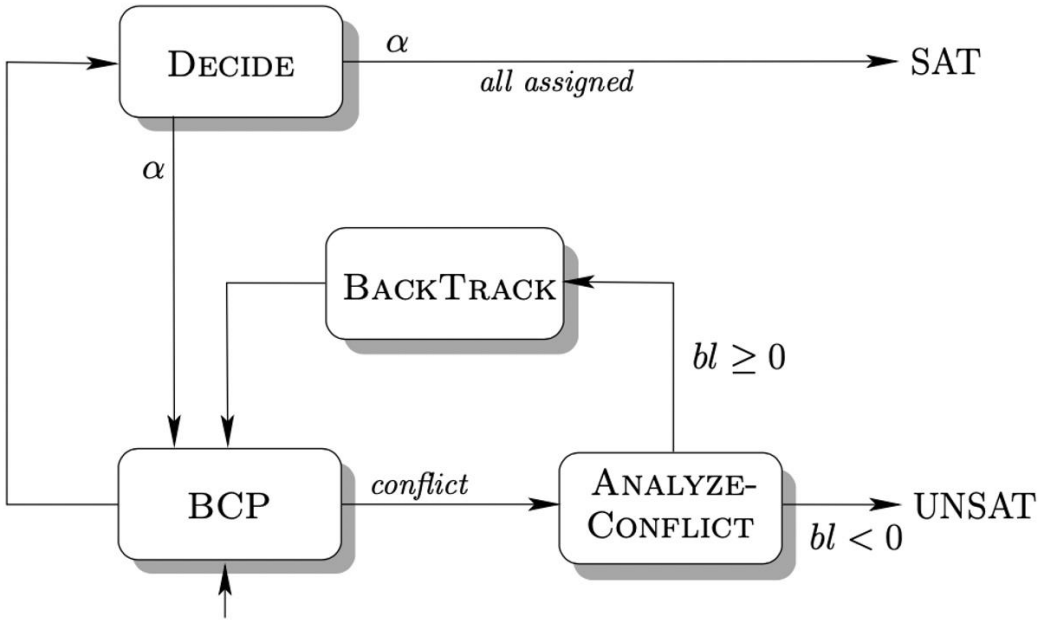


Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
1  dl ← 0;                    //decision level
2  if UnitPropagation(F,α)==CONFLICT then return UNSAT
3  while ∃ unassigned variables do
       /* PickBranchVar picks a variable to assign and
          picks the respective value                    */
4      (x, v) ← PickBranchVar(F, α);
5      dl ← dl + 1;
6      α ← α ∪ {(x, v)};
7      if UnitPropagation(F,α)==CONFLICT then
8          bl ← ConflictAnalysis(F, α);
9          if bl < 0 then
10             return UNSAT;
11         else
12             BackTrack(F, α, bl);
13             dl ← bl;
14 return SAT;
```

CDCL – Algorithm

- Analyze-Conflict : non-chronological backtracking + clause learning + vivification
- Decide : Branching strategy and phasing strategy



Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
1  dl ← 0;                    //decision level
2  if UnitPropagation(F,α)==CONFLICT then return UNSAT
3  while ∃ unassigned variables do
       /* PickBranchVar picks a variable to assign and
          picks the respective value
4      (x, v) ← PickBranchVar(F, α);
5      dl ← dl + 1;
6      α ← α ∪ {(x, v)};
7      if UnitPropagation(F,α)==CONFLICT then
8          bl ← ConflictAnalysis(F, α);
9          if bl < 0 then
10             return UNSAT;
11         else
12             BackTrack(F, α, bl);
13             dl ← bl;
14 return SAT;
```

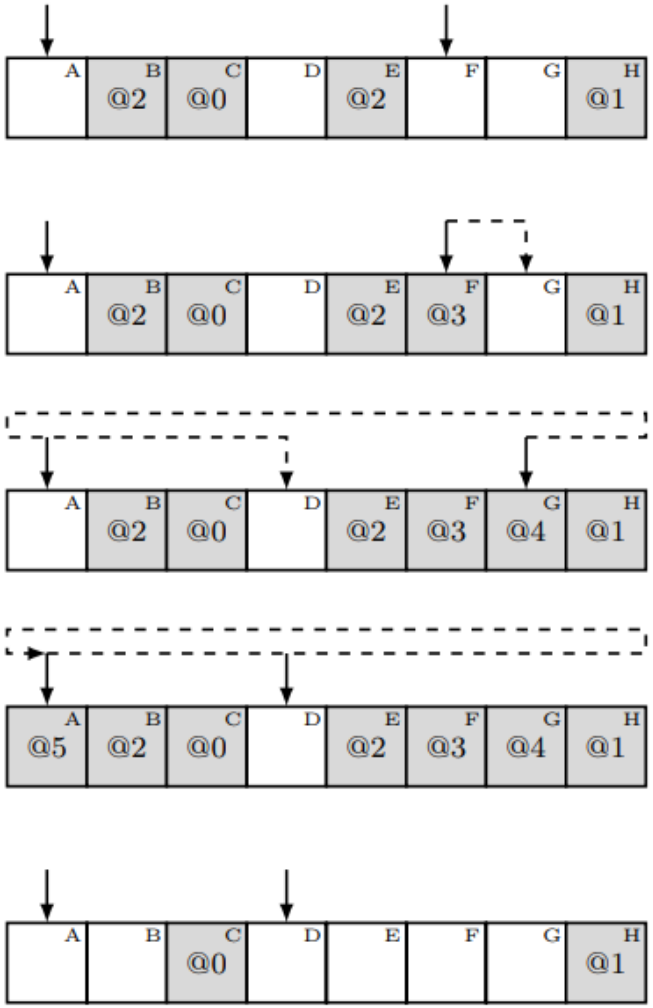
- Clause learning
- Clause management
- Lazy data structures
- Restarting
- Branching
- Phasing
- Mode Switching
- …

CDCL – Lazy data structure

[ZhangStickel' 00] [MoskewiczMadiganZhaoZhangMalik' 01]

Efficient UP: 2 watched literals

- In each non-satisfied clause "watch" two non-false literals
- For each literal remember all the clauses where it is watched



CDCL – Branching heuristics

- Static branching heuristic: e.g. Ordered BDDs
- Dynamic branching heuristic considering current partial assignment.
  - dynamic literal individual sum heuristic (DLIS)
- Dynamic branching heuristic considering learning clauses
  - Variants of DLIS
  - Variable state independent decaying sum(VSIDS) and its variants
    - Normalized VSIDS(NVSIDS) : exponential moving average
    - Exponential VSIDS(EVSIDS) : proposed by MiniSAT
    - Literal state independent decaying sum(LSIDS)
  - Variable move to front(VMTF) [Ryan Thesis 2004]
  - Average conflict-index decision score(ACIDS)
- Reinforcement learning based branching heuristic: multi-armed bandit(MAB)
  - Conflict history-based branching(CHB): $(1-\alpha)s + \alpha \cdot r, r = \frac{multiplier}{nConf - lastConf_v + 1}$
  - Learning rate based branching(LRB)
- Dynamic switching between multiple heuristics
  - Kissat-MAB switching between CHB and VSIDS by Upper Confidence Bound(UCB)

CDCL – Branching heuristics

- Branching heuristics are used for deciding which variable to use when branching.
  - Nowadays, solvers prefer the variable which may cause conflicts faster.
- Variable State Independent Decaying Sum (VSIDS) [MoskewiczMadiganZhaoZhangMalik'01]
  - **Compute score for each variable, select variable with highest score**
  - Initial variable score is number of literal occurrences.
  - For a new conflict clause c: score of all variables in c is incremented.
  - Periodically, divide all scores by a constant. % forgetting previous effects

CDCL – Branching heuristics

- Most popular: the exponential variant in MiniSAT (EVSIDS)
  - The scores of some variables are *bumped* with *inc*, and *inc* decays after each conflict.
  - Initialize *score* to 0, the bump score *inc* default to 1.
  - *inc* multiply 1/*decay* after each conflict, *decay* initialized to 0.8, increased by 0.01 every [5k] conflicts, the maximum of *decay* is 0.95.
  - The score of variables in conflict clause *c* are bumped with *inc*.

  + the *score* of the variables related with this conflict analysis are bumped with *inc*/2 [reason side bump used in Maple].

CDCL – Branching heuristics

- Learning rate based branching heuristic(LRB)[LiangGaneshPoupartCzarnecki,16SAT]
  - Compute score A_v for each variable v, select variable with highest score.

$$A_v = (1-\alpha)A_v + \alpha \cdot \left(\frac{\mathrm{P(v,I)}}{\mathrm{L(I)}} + \frac{\mathrm{A(v,I)}}{\mathrm{L(I)}}\right)$$

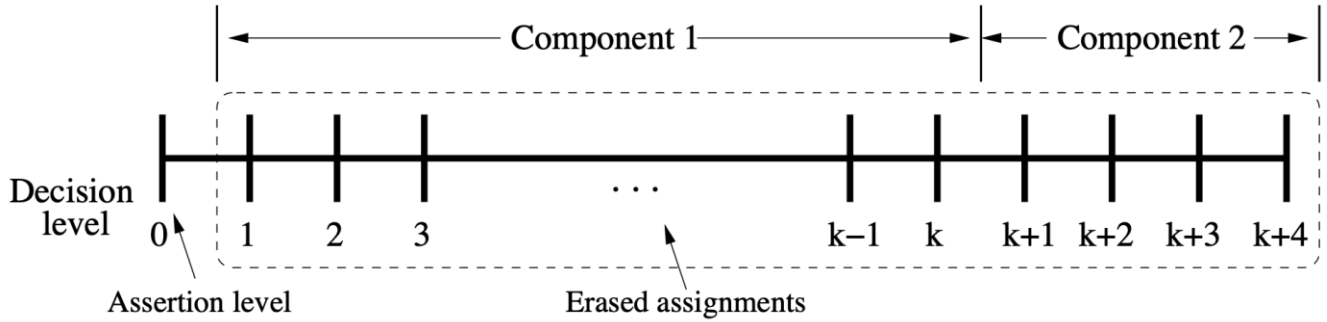
    - $\frac{\mathrm{P(v,I)}}{\mathrm{L(I)}}$ and $\frac{\mathrm{A(v,I)}}{\mathrm{L(I)}}$ are called learning rate and reason side rate.
    - I is the interval of time between the assignment of v until v transitions back to being unassigned.
    - P(v,I) is the number of learnt clauses in which v participates during I.
    - V(v,I) is the number of learnt clauses which v reasons in generating in I.
    - L(I) is the number of learnt clauses generated in I.

- VMTF [Ryan Thesis 04]
  - Using a priority queue for select variables.
  - Move the variables in the conflict clause to the front of the queue.

CDCL – Phase Saving

Phase saving [PipatsrisawatDarwiche,07SAT]

- Phase : the value which variable should assign when branching.
- Phase saving (progress saving): Save assignments when backtracking.
  - returns the phase of a variable x corresponding to the last time x was assigned.
- Reason why phase saving: Avoid too many useless erasures and decisions.
  - reusing the trail can reduce the cost of restarts [RamosVanDerTakHeule,11JSAT]

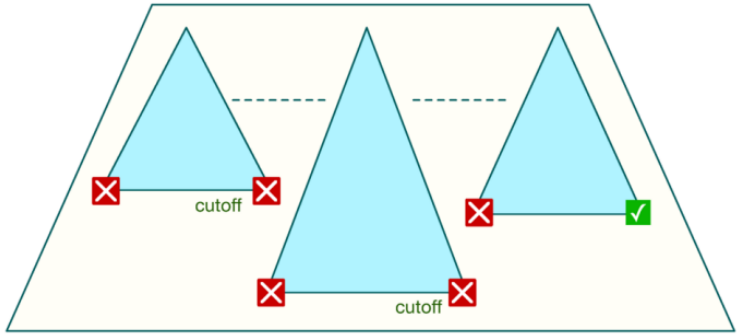


CDCL – Learnt Clause Removal

- Reason why **Clause Database Reduction**:
  - Not all of them are helpful;
  - UP gets slower with memory consumption.
- Measurement criteria of clauses:
  - least recently used (LRU) heuristics: discard clauses not involved in recent conflict clause generation
  - **Literal Block Distance(LBD): number of distinct decision levels in learnt clauses**, proposed in glucose. [AudemardSimon,09IJCAI]
  - 3-tired clause Learned clause management : *core* are clauses with LBD≤ 3; *mid_tire* retain recently used clause with LBD up to 6; *local* saving other clauses. [Chanseok Oh, 15SAT]
- Reduction Method in 3-tier method:
  - *core* never be removed;
  - Periodically remove half *local* clauses based on score.
  - Periodically move some recently not used clauses in *mid_tire* to *local*.
  - move clauses encounter in *local* many times to *mid_tire*, and same from *mid_tire* to *core*.

CDCL – Effective Restart

- Periodical traceback to 0 decision level.
  - clause learning and search restarts correspond to a proof system as powerful as general resolution, and stronger than DPLL proof system; practically effective.

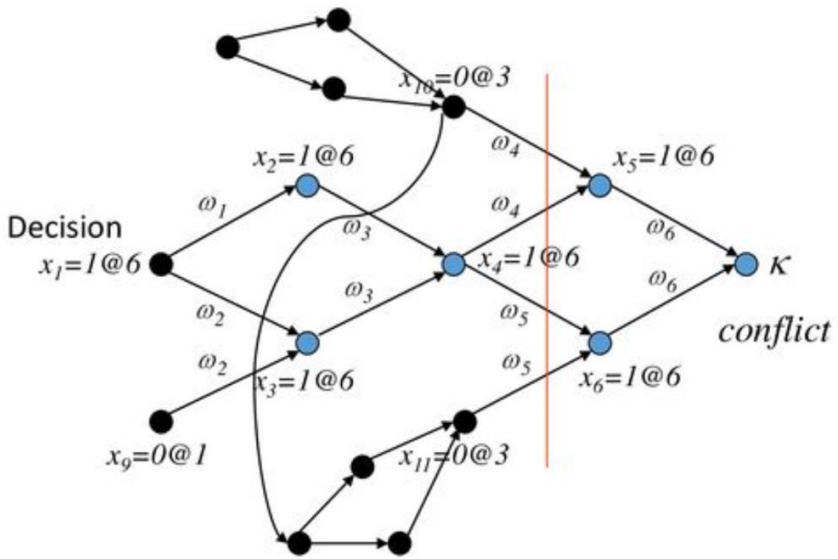


- Restart policies:
  - Luby series: 1 1 2 1 1 2 4 1 1 2 1 1 2 4 8 …
  - Glucose restart (rapid) : When average LBD of some current learnt clauses is great than the average LBD of all learnt clauses. [AudemardSimon,12CP]
- A conjecture: rapid restarts generally helps deriving a refutation proof, while remaining in the current branch increases the chance of reaching a model
  - interleave “stabilizing” mode (no restarts) and “focused” mode [Chanseok Oh,15SAT]

CDCL – Clause Simplification

- Remove some literals which can be conducted by another literal in the clause.

- $reason(x_3) = \omega_4, reason(x_6) = \omega_5, \ldots$
- Local / General implication graph



Learnt Conflict clause: ($x_{10}, \neg x_4, x_{11}$)

$\overline{x_{10}} \rightarrow \overline{x_{11}}$

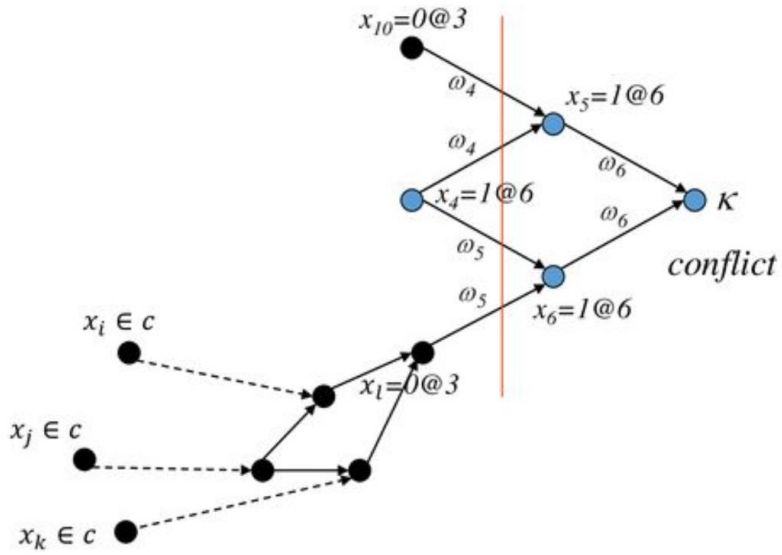
or, $x_{11} \rightarrow x_{10}$

Clause minimization: Drop x_{11}

CDCL – Clause Simplification

- Remove some literals which can be conducted by other literals in the clause.

  - More generally: if we have a c.c. $c = (x_0 \vee \cdots \vee x_n)$, and
  - ... all reverse paths on the implication graph hit c literals...



$$\overline{x_i} \wedge \overline{x_j} \wedge \overline{x_k} \rightarrow \overline{x_l}$$

or, $x_l \rightarrow (x_i \vee x_j \vee x_k)$

Clause minimization: Drop x_l

SAT solving – Others

- Preprocessing / Inprocessing (Interleave search and preprocessing)
  - Bounded Variable Elimination
  - Variable Elimination with “AND” Gates
  - Blocked Clauses
- Parallel SAT Solving
  - Divide and Conquer – explicit search space partitioning
  - Cube and Conquer – implicit load balancing
  - Diversify and Conquer – portfolio search
- Portfolios
  - Pure portfolios
  - Portfolios with Clause Learning
- Incremental SAT Solving

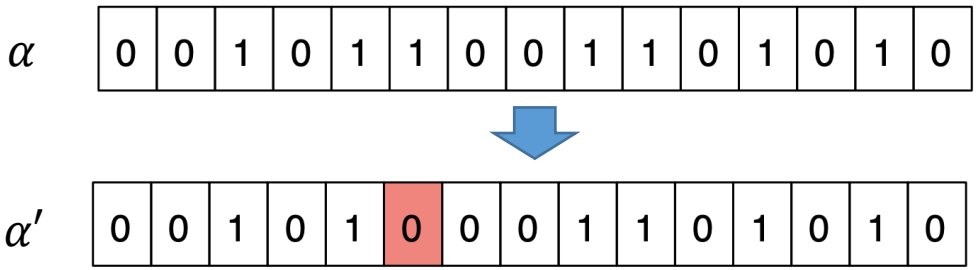
Outline

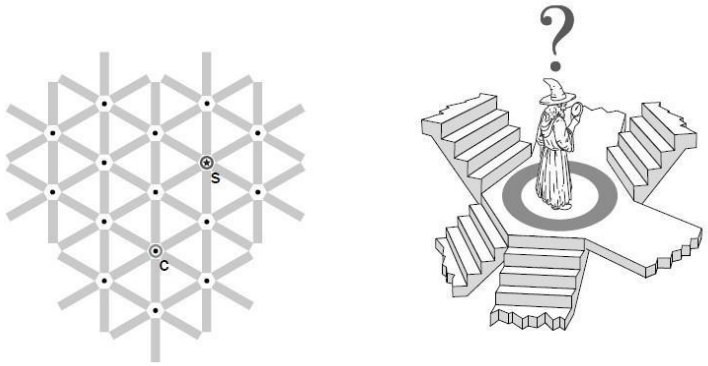
- SAT Basis
- SAT Encoding
- CDCL
- Local Search
- Hybrid SAT Solving

Local Search - Basis

Stochastic local search (**SLS**) for SAT

- Begin with a complete assignment
- Iteratively modify the assignment by **flipping** a variable picked by heuristics.





Geometrical view: as a walk in the space of 2^n assignment

Local Search - Basis

search space S

(SAT: set of all complete truth assignments to propositional variables)

solution set $S' \subseteq S$

(SAT: models of given formula)

neighbourhood relation $N \subseteq S \times S$

(SAT: neighbouring variable assignments differ in the truth value of exactly one variable)

evaluation function g : $S \rightarrow R^+$

(SAT: number of clauses unsatisfied under given assignment)

Local Search - Basis

Algorithm : Local Search Framework for SAT

```
1 begin
2     α ← a complete assignment;
3     while not reach terminal condition do
4         if α satisfies F then return α;
5         pick a variable x;
6         α ← α with x flipped;
7     return "Solution not found";
```

Scoring functions

Search strategies

Run a small example

- Neighbourhood relation: two assignments are neighbors if and only if they differ in the truth value of exactly one variable

$F=\{x_1 \vee \sim x_2,\ x_1 \vee x_2,\ x_2,\ \sim x_1 \vee x_2 \vee \sim x_3\}$		
	assignment	unsatisfied clauses
S	000	$x_1 \vee x_2,\ x_2$

- S=<000>, N(S)={S1,S2,S3}={<100>,<010>,<001>}
- g(S)=2
- g(S1) =1
- g(S2) =1
- g(S3) = 2

Score of variables

- Instead of using evaluation functions on assignments, we usually define scoring functions for variables.
- Under assignment S, score(x) = g(S)-g(S'), where S' differs from S only in the value of x. This is a scoring function of variables.

$F=\{x_1 \vee \sim x_2,\ x_1 \vee x_2,\ x_2,\ \sim x_1 \vee x_2 \vee \sim x_3\}$		
	assignment	unsatisfied clauses
S	000	$x_1 \vee x_2,\ x_2$

- $score(x_1)=g(000) - g(100)=2-1=1$
- $score(x_2)=g(000) - g(010) = 2-1=1$
- $score(x_3)=g(000) - g(001) = 2-2=0$

Iterative improvement

Invariant of Iterative Improvement for SAT

GSAT [Selman et al, AAAI 1992]

- S := a random complete assignment;
- while (!termination condition)
    - if (S is a solution) return S;
    - x := a variable with the best score;
    - S := S with x flipped;
- return S;

Focused random walk (WalkSAT)

- In this type of random walk step, first a random unsatisfied constraint *c is selected*.
- *Then, one of the* variable appearing in *c is randomly selected and flip (thus forces c to become satisfied)*.

```
procedure WalkSAT (F, maxTries, maxSteps, slc)
    input: CNF formula F, positive integers maxTries and maxSteps,
        heuristic function slc
    output: model of F or 'no solution found'
    for try := 1 to maxTries do
        a := randomly chosen assignment of the variables in formula F;
        for step := 1 to maxSteps do
            if a satisfies F then return a end
            c := randomly selected clause unsatisfied under a;
            x := variable selected from c according to heuristic function slc;
            a := a with x flipped;
        end
    end
    return 'no solution found'
end WalkSAT
```

Clause weighting for SAT

- Date back to the Breakout method (1993): increases the weight of each unsatisfied clause by one when reaching local optima.

Modern clause weighting usually have a "smoothing" mechanism to decrease weights.

- discrete Lagrangian method (DLM): DLM follows Breakout's weight increment scheme, but additionally decrements clause weights by a constant amount after a fixed number of increases;
- pure additive weighting scheme (PAWS)[Thornton+05]
  - PAWS updates clause weights in local optima as follows. First, the clause weights of all unsatisfied clauses are increased by one; then, all clause weights are decreased by one after a fixed number of increases.[]
- the scaling and probabilistic smoothing (SAPS) [HutterHoos+,02]
  - when reaching a local optimum, with some probability $w(c) = \rho w(c) + (1 - \rho)\overline{w}$
- Smoothed Weighting based on Threshold (SWT)[CaiSu,13]
  - When $\overline{w}$ reaching a threshold, smooth the weights by $w(c) = \rho w(c) + q\overline{w}$

Scoring functions

A basic Scoring Function

- A common scoring function for SAT, which is named ‘score’.
- Under assignment S, score(x) = cost(S)-cost(S’), where S’ differs from S only in the value of x, cost(S) is the number of unsatisfied clauses under S.
- Score(x)=make(x)-break(x)

- Efficient implementation: caching or non-caching, depends..

Scoring functions

A Scoring Function can be:

- a property of the variable, such as score, age, frequency …
- any mathematical expression with one or more properties.

Other Scoring functions

- age(x) = the number of steps since x has changed value
- frequency(x) = a count on how many times x changes its value
- wscore(x) = the weighted version of score, using clause weighting techniques
- Score(x)+age(x)/T, where T is a parameter
- $A^{score(x)}$
- $score(x)^{B}$
- …

Dynamic Scoring functions

- Change the parameters or the expression of the scoring function during the search

Scoring Functions - Probability based scoring functions

- Sparrow (2010) [BalintFröhlich,2000SAT]
- probSAT(2011) [Balint Schöning:,2001SAT]
  - Focused random walk algorithm
  - In each step, probSAT computes f(x) for each variable x in the clause for a given scoring function f(·),
  - and then chooses a random variable x according to probability $\sum f(x)\ z \in C\ f(z)$.
  - several functions f(·) for probSAT, mainly including exponential and polynomial functions of the break property.

Scoring functions - Satisfaction degree

- Given an assignment S=$\{x_1 =1, x_2 =0, x_3 =0, x_4 =1, x_5 =1\}$
- c1=$x_1 \vee x_2 \vee \neg x_3 \vee x_4 \vee \neg x_5$
- c2=$x_1 \vee \neg x_2 \vee x_3 \vee \neg x_4 \vee \neg x_5$

Both clauses are satisfied.

But c1 is a 4-satised clause, while c2 has 1-satised.

1-satised clauses are the most endangered satisfied clauses. →critical clauses

Scoring functions - Second level scoring functions

- Second Level Scoring Functions [Cai+, 13AAAI/AIJ]
  - $make_2$(x) is the number of 1-satifised clauses that would become 2-satised by flipping x.
  - $break_2$(x) is the number of 2-satifised clauses that would become 1-satised by flipping x.
  - $score_2(x) = make_2(x) - break_2(x)$

- Use $score_2$
  - Break ties
  - Hybrid scoring functions

Analysis on second level score

- Proposition: For a random 3-SAT formula F(n,m), under any solution s to the formula, the number of 1-satised clauses is more than m/2.
- Proof: Since half literals are positive and half are negative, for any complete assignment, the number of true literals is half of all literals $T(\alpha) = \frac{mk}{2}$ (k=3)
- Now we calculate T(α) in another way, by adding up true literals in the i-satisfied clauses(0≤i ≤k)

$$T(\alpha) = \sum_{i=0}^{k} i m_i = \sum_{i=1}^{k} i m_i$$

$$= m_1 + \sum_{i=2}^{k} i m_i$$

$$\geq m_1 + 2\sum_{i=2}^{k} i m_i$$

$$= m_1 + 2(m - m_1)$$

$$= 2m - m_1$$

Analysis on second level score

Together, we have

$$\frac{mk}{2} \geq 2m - m_1$$

Which yields a lower bound of the number of 1-satisfied clauses as

$$m_1 \geq \left(2 - \frac{k}{2}\right) m$$

Suitable for formulas with long clauses (k>3).

The cycling problem of local search

- Cycling problem, i.e., revisiting candidate solutions

- A key factor to bad performance
  - wastes time
  - prevents it from getting out of local minima

- Cycling is an inherent problem of local search
  - local search does not allow to memorize all previously visited parts of the search space.

Methods to deal with cycling

- Naive methods
  - Random walk
  - Non-improving search
  - Restart

- The tabu mechanism
  - forbids reversing the recent changes, where the strength of forbidding is controlled by a parameter called tabu tenure [Glover, 1989].

- Configuration checking [CaiSu 2011, 2012]
  - Initially for vertex cover, then SAT/MaxSAT, among many others
  - Considers the circumstance of the variables, a variable is allowed to flip if its circumstance has changed since its last flip.

Tabu for SAT

- An FIFO queue: tabuList[]
- Each step
  - a variable x not in tabuList is chosen to flip the value
  - Add x to tabuList
  - If (tabulist.size > tt) remove the first element of tabuList

- Think: how to check whether a variable is tabu, by the age of a variable (so that we do not need tabuList)? (age(x): the number of steps since the last time x changed its value).

Tabu for SAT

Note: Cycles of length at most m can be prevented by tabu mechanism with tabu tenure tt=m.

Trade-off of choosing tt:

- tt too low -> fail to prevent cycling
- tt too high -> an excessive restriction of neighborhoods

Advanced TS methods:

- **Tabu with Aspiration**
  a variable can be chosen if its score is very large, regardless of whether it is forbidden by Tabu strategy;
- **Robust Tabu Search** [Taillard, 1991]:
  repeatedly choose tt from given interval;
- **Reactive Tabu Search** [Battiti and Tecchiolli, 1994]:
  dynamically adjust tt during search;

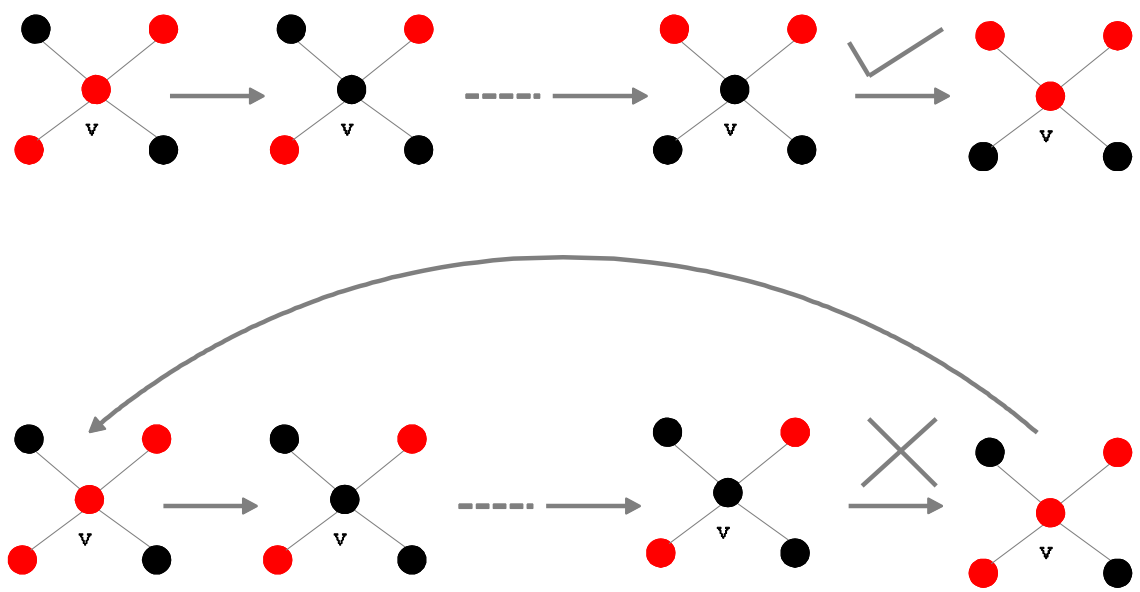
Configuration Checking (CC)

- Address cycling problem by Configuration Checking (CC) [2011].

- CC is found effective for the following types of problems:
  - Assignment Problems: to find an assignment to all variables such that satifises the constraints (and optimized).
  - Subset Problems: to find a subset from a universe set such that satisfies the constraints (and optimized).

A simple CC for SAT

- $\boldsymbol{N(x) = \{y | y\ and\ x\ occur\ in\ at\ least\ one\ clause\}}$
- **configuration**: the configuration of a variable x is a vector C_x consisting of truth value of all variables in $N(x)$ under current assignment s (i.e., $C_x = s|_{N(x)}$).
- **A simple CC for SAT**: if the configuration of x has not changed since x's last flip, then it should not be flipped.



The Use of CC

- Use CC
  - to filter candidate variables
  - to give preference to CC variables

- Used in many successful local search SAT and MaxSAT algorithms.

Naïve Implementation of CC

- An accurate implementation of CC
  - Store the configuration (i.e., truth values of all its neighbors) for a variable x when it is flipped
  - Check the configuration when considering flipping a variable

- For a formula F, let $\Delta(F) = \max\{\#N(x): x \in V(F)\}$
- It needs $O(\Delta(F))$ for both storing and checking the configuration for a variable.
- Thus, the worst case complexity of CC in each step is $O(\Delta(F)) + O(\Delta(F)n)$

Efficient Implementation of CC

- Observation: when a variable is flipped, the configuration of all its neighboring variables has changed.

- Efficient Implementation:
- Auxiliary data structure --- CC array
  - CC[x] = 1 means the configuration of x has been changed since x's last flip;
  - CC[x] = 0 on the contrary.

- Maintain the CC array
  - Rule 1: In the beginning, for each variable x, CC[x] is initialized as 1.
  - Rule 2: When flipping x, CC[x] is reset to 0, and for each $y \in N(x)$, CC[y] is set to 1.

Efficient Implementation of CC

- Complexity of the approximate implementation
  - O(1) for checking whether a variable is configuration changed (check whether CC[x]=1).
  - update CC values for N(x).
  - Thus, the worst case complexity of CC in each step is
    $O(n) + O(\Delta(F))$

- Indeed, the number of candidate variables for flipping is much smaller than n.

On the analysis of CC

- When it works? When it does not work?
- The effectiveness of CC is related to the neighborhood of variables.

For a random k-SAT formula $F_k(n, m)$, we fix an arbitrary variable x, calculate $\mathbb{E}(\#N(x))$.

For any clause c, $E(c)$: "x and y both appear in c", then

$$p = Pr(E(c)) = \binom{n-2}{k-2} / \binom{n}{k} = \frac{k(k-1)}{n(n-1)}$$

Hence,

$$Pr(y \in N(x)) = 1 - Pr(y \notin N(x)) = 1 - (1-p)^m$$

Let I_y be the indicator variable for the event $\{y \in N(x)\}$,

$$I_y = \begin{cases} 1, & \text{if } y \in N(x) \\ 0, & \text{if } y \notin N(x) \end{cases}$$

$$\mathbb{E}(I_y) = Pr(y \in N(x)) = 1 - (1-p)^m$$

On the analysis of CC

The expectation of the size of $N(x)$ can be obtained as following

$$\mathbb{E}(\#N(x)) = \mathbb{E}(\sum_{y \in V(F)\setminus\{x\}} I_y) \quad (1)$$

$$= \sum_{y \in V(F)\setminus\{x\}} \mathbb{E}(I_y) \quad (2)$$

$$= (n-1)(1-(1-p)^m) \quad (3)$$

$$\approx (n-1)(1-(\frac{1}{e})^{pm}) \quad (4)$$

$$= (n-1)\left(1-(\frac{1}{e})^{\frac{k(k-1)}{n(n-1)}m}\right) \quad (5)$$

$$= (n-1)\left(1-(\frac{1}{e})^{\frac{k(k-1)r}{(n-1)}}\right) (let \quad r = m/n) \quad (6)$$

On the analysis of CC

Using the limit $\lim_{N\to+\infty}(1-e^{-\frac{c}{N}})=\frac{c}{N}$ (where c is a constant), and let $N=n-1$, $c=k(k-1)r$, we have

$$\lim_{n\to\infty}\mathbb{E}(\#N(x))=(n-1)\left(1-(\frac{1}{e})^{\frac{k(k-1)r}{(n-1)}}\right) \tag{7}$$

$$=(n-1)\frac{k(k-1)r}{n-1} \tag{8}$$

$$=k(k-1)r \tag{9}$$

On the analysis of CC

Now, by using the inequality $(1-\frac{1}{n})^n < \frac{1}{e}$ $(n > 1)$, we have

$$\mathbb{E}(\#N(x)) = (n-1)(1-(1-p)^m) \tag{10}$$

$$> (n-1)(1-(\frac{1}{e})^{pm}) \tag{11}$$

$$= (n-1)\left(1-(\frac{1}{e})^{\frac{k(k-1)r}{(n-1)}}\right) \tag{12}$$

$$= (n-1) - \frac{n-1}{e^{\frac{k(k-1)r}{(n-1)}}} \tag{13}$$

When $n-1 < e^{\frac{k(k-1)r}{(n-1)}}$, or equivalently, $ln(n-1) < \frac{k(k-1)r}{(n-1)}$, we have $\mathbb{E}(\#N(x)) > (n-1)-1 = n-2$.

$\rightsquigarrow$ most variables have all other variables as their neighbourhood

$\rightsquigarrow$ the CC strategy almost degrades to the tabu with tt=1.

When CC Becomes Ineffective on random k-SAT

$f(n) = ln(n-1) - \frac{k(k-1)r}{n-1}$ is a monotonic increasing with n $(n > 1)$.
$f(n) < 0$ iff $n \leq \lfloor n^* \rfloor$, where n^* is a real number s.t. $f(n^*) = 0$.

Formulas	3-SAT ($r = 4.2$)	4-SAT ($r = 9.0$)	5-SAT ($r = 20$)	6-SAT ($r = 40$)	7-SAT ($r = 85$)
n^*	11.652	32.348	90.093	223.095	564.595

Table: The n^* value such that when $n < n^*$, the size of any variable's neighborhood is bigger than $n-2$, and the CC strategy degrades to the tabu method with tabu tenure 1.

Variants of CC for SAT

- The typical CC strategy for SAT is Neighboring Variables based CC.

- We can have different CC variants by defining different configuration and checking methods.
  - In Clause States based CC (CSCC), the configuration of a variable x is a vector that consists of the states of all the clauses in which x appears.
  - Quantitative CC, the CC value is an integer.
  - Dynamic Threshold CC, a dynamic checking mechanism
  - Double CC, combining NVCC and CSCC

Local Search Solver - CCAnr

- Configuration checking (CC)
- Smoothed Weighting (SW)
- Aspiration
- Score: weighted version
- configuration changed decreasing(CCD): score>0 and configuration is modified.
- significant decreasing(SD) : score > $\overline{w}$

Algorithm 5: generalized *pickVar*-heuristic CCA

```
1 //greedy mode
2 if CCD variables exist then return a CCD variable with the greatest score;
3 if SD variables exist then return an SD variable with the greatest score;
4 //diversification mode
5 update clause weights;
6 select a random falsified clause c;
7 return a variable in c;
```

1. the tie-breaking mechanism in the greedy mode: CCAnr break ties by favoring the oldest variable in the greedy mode, as Swcca does.

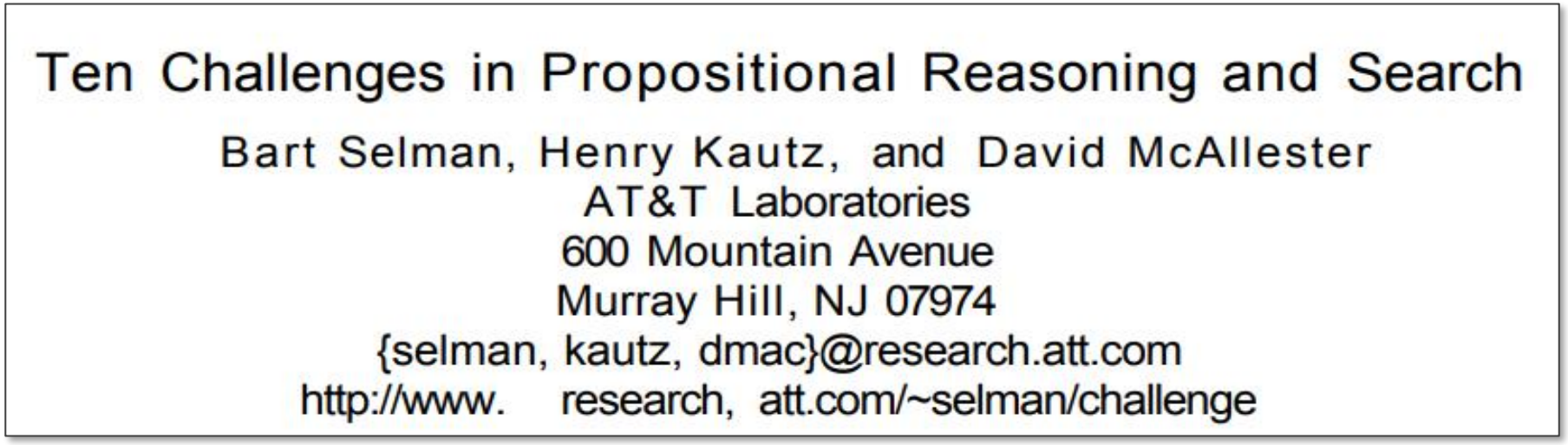
2. the clause weighting scheme: CCAnr adopts a Smooth Weighting based on Threshold (SWT) scheme. Each time the SWT weighting is called, clause weights of all falsified clauses are increased by one; further, if the averaged weight $\overline{w}$ exceeds a threshold γ, all clause weights are smoothed as $w(c_i) := \lfloor \rho \cdot w(c_i) \rfloor + \lfloor q \cdot \overline{w} \rfloor$.

3. the pick-var heuristic in the diversification mode: CCAnr selects the variable with the greatest *score* from an falsified clause, breaking ties by favoring the oldest one.

Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search
- Hybrid SAT Solving

Hybrid Solving – The 7th Challenge of SAT



Ten Challenges in Propositional Reasoning and Search

Bart Selman, Henry Kautz, and David McAllester
AT&T Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
{selman, kautz, dmac}@research.att.com
http://www. research, att.com/~selman/challenge

Challenge 7: Demonstrate the successful combination of stochastic search and systematic search techniques, by the creation of a new algorithm that outperforms the best previous examples of both approaches.

---**AAAI 1997** , Bart Selman, Henry Kautz and David McAllester

Hybrid Solving – Related works

- Use a local search solver as the main body solver.
  - ➢hybridGM （SAT 2009）, SATHYS （LPAR 2010）
  - ➢GapSAT: use CDCL as preprocessor before local search (SAT 2020)
  - ➢Use resolution in local search (AAAI 1996, AAAI 2005)
- DPLL/CDCL as the main body solver
  - ➢HINOTOS: local search finds subformulas for CDCL to solve (SAT 2008)
  - ➢WalkSatz: calls WalkSAT at each node of a DPLL solver Satz (CP 2002)
  - ➢CaDiCaL and Kissat: a local search solver is called when the solver resets the saved phases and is used only once immediately after the local search process (2019)
- Sequential call local search and CDCL
  - ➢Sparrow2Riss, CCAnr+glucose, SGSeq (SAT Competitions 2014,2015)

Hybrid Solving – A turning point

It has been a long belief that SLS is good at solving random formulas, while CDCL is powerful at solving structured formulas.

The emerging modern SLS solvers, particularly Sattime (Li, 11SAT), probSAT(BalintSchoning,11SAT), CCAnr (CaiLuoSu,15SAT), YalSAT (Biere,17),show that SLS can be competitive on hard combinatorial instances.

Exapmles of CCA solvers on solving structured benchmarks.

- a variant of CCASat used in FCC projects, solving more SAT instances then CDCL solvers (Kevin Brown et al, AAAI 2016, PNAS 2017),
- CCAnr showed good performance in software testing benchmark from Microsoft (Armin Biere, SAT 2015)
- specified SLS solver in matrix multiplication (Marijn, SAT 2009),
- CCAnr solving more instances from PTN problem (CaiZhangLuo, CP 2021) than state of the art CDCL solvers except one .
- …

A Quest on efficient hybrid solvers

1. Sequential calling SLS and CDCL solvers

SAT

Hard combinatorial		
Gold	Silver	Bronze
SparrowToRiss	CCAnr+glucose	SGSeq

From SAT Competition 2014

From Sparkle SAT Challenge 2018

Rank	Solver	rel. marginal contribution	PAR2 (stand-alone)	Note
1	CryptoMiniSatv5.5	**12.97%**	4740.02	
2	ReasonLS	**9.68%**	4775.40	
3	minisat-2.2.0_PADC	**9.07%**	5925.19	
4	glucose-3.0_PADC	**8.41%**	6065.19	
5	UPLS	**8.18%**	8569.31	
6	Riss7	**7.81%**	5336.48	
7	probSAT	**6.99%**	9041.22	
8	CaDiCaL	**6.93%**	5372.57	
9	glu_mix	**6.20%**	5668.08	
10	BreakIDGlucoseSEL	**5.42%**	5702.63	

A Quest on efficient hybrid solvers

UP based Initialization

2. Trying to utilize reasoning to improve SLS

Thinking:

The power of CDCL mainly comes from reasoning techniques: Unit Propagation, Clause learning

Attempts:

Using unit propagation

- during local search
- initialization√

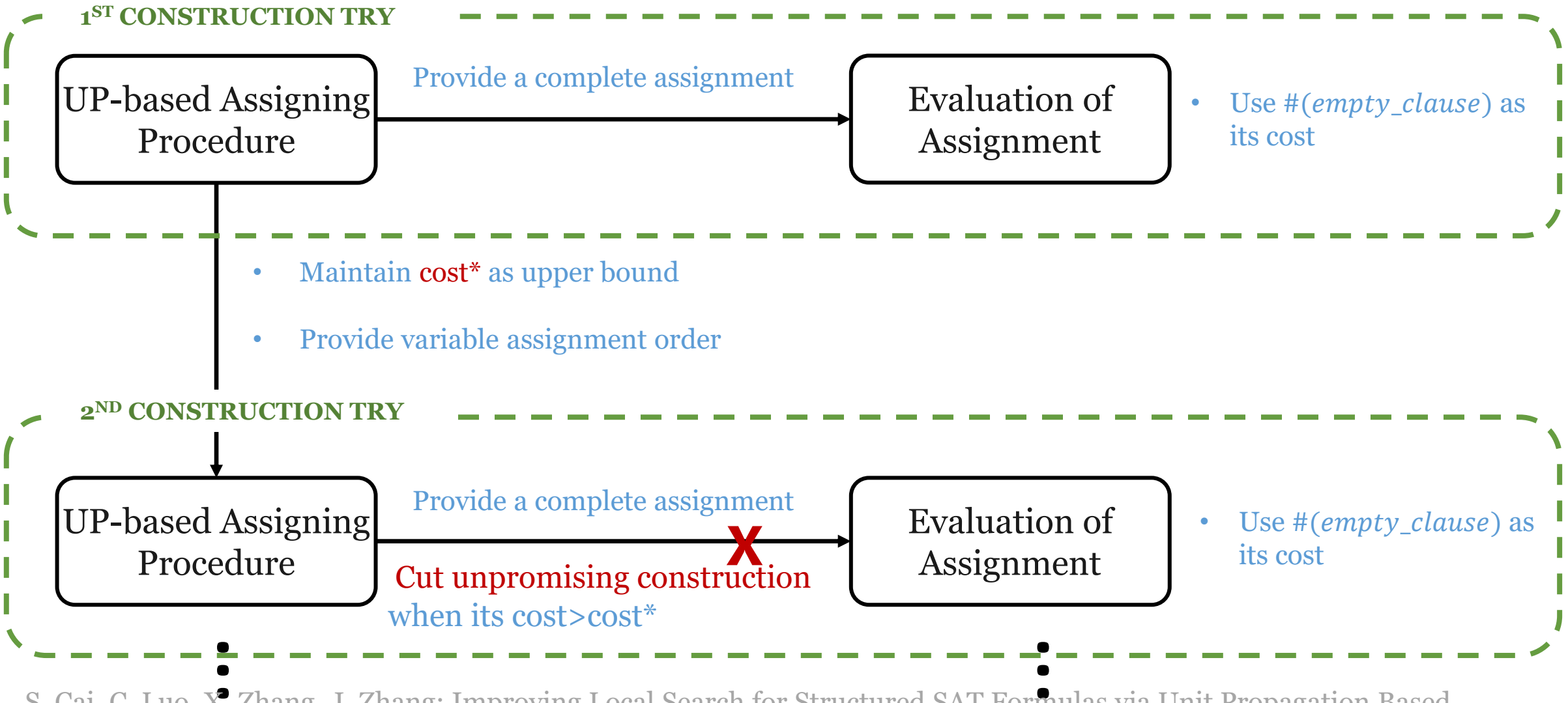
Using resolution

- during search

[UP]decision[UP]decision…. → complete assignment

A Quest on efficient hybrid solvers

UP based construct-and-cut Initialization



S. Cai, C. Luo, X. Zhang, J. Zhang: Improving Local Search for Structured SAT Formulas via Unit Propagation Based Construct and Cut Initialization (Short Paper). CP 2021: 5:1-5:10

A Quest on efficient hybrid solvers

UP based construct-and-cut Initialization

- Improving SLS in solving some mathematical and industrial benchmarks

Table 1 Results of local search solvers and CnC-enhanced local search solvers on all benchmarks.

Benchmark	*CCAnr*		*CCAnr+cnc*		*ProbSAT*		*ProbSAT+cnc*		*Sattime*		*Sattime+cnc*		*YalSAT*	
	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2
FCC (9879)	7878	2091.6	**8110**	**1868.2**	5407	4577.7	5477	4506.5	7054	2911.8	7078	2900.0	7136	2881.1
PTN (23)	13	4718.0	**23**	**127.0**	5	7885.0	20	2161.7	9	6790.7	18	2945.3	14	4490.3
SN5 (6)	2	7364.5	**4**	**4969.5**	0	10000.0	0	10000.0	0	10000.0	1	8708.7	0	10000.0

Table 2 Results of *CCAnr+cnc* and its CDCL competitors on all benchmarks.

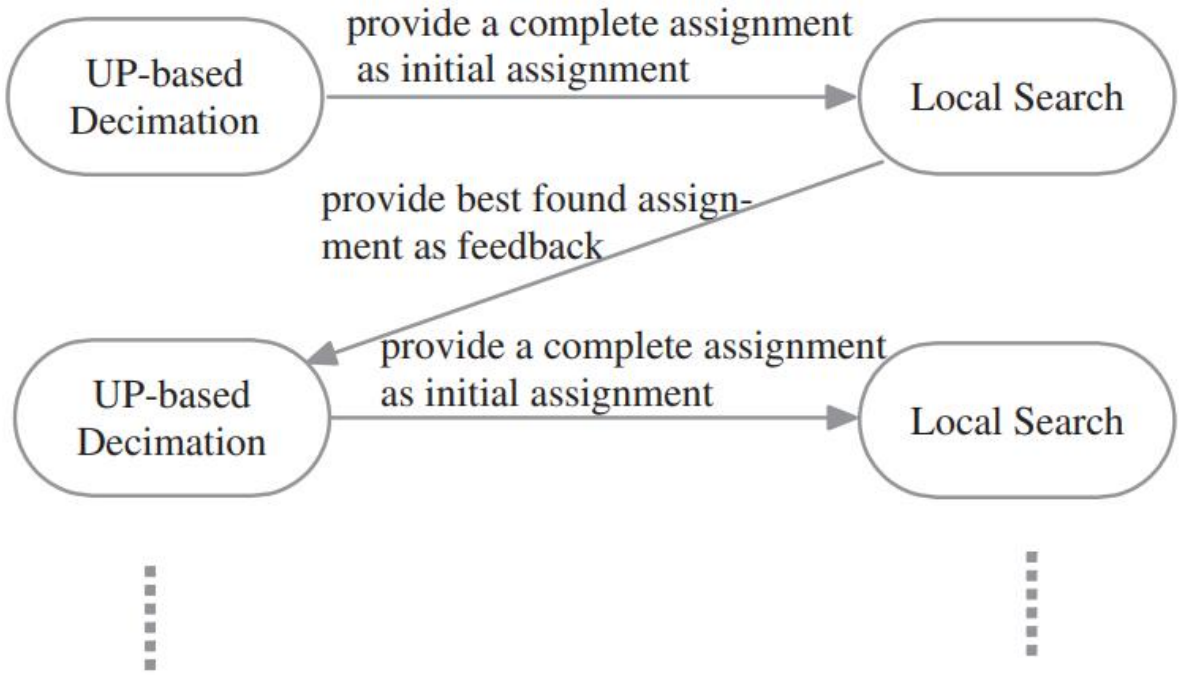
	CCAnr+cnc		*CaDiCaL*		*CaDiCaL_sat*		*Maple_LCM_Dist*		*MapleCOMSPS*		*Kissat*		*Kissat_sat*	
	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2	#SAT	PAR2
FCC (9879)	8110	1868.2	7674	2326.9	7783	2211.9	7788	2183.2	7783	2183.0	7949	2042.8	**8163**	**1819.1**
PTN (23)	**23**	**127.0**	17	3274.2	17	3007.4	0	10000.0	1	9639.0	19	2215.7	21	1402.5
SN5 (6)	**4**	**4969.5**	0	10000.0	0	10000.0	0	10000.0	0	10000.0	0	10000.0	1	9130.7

S. Cai, C. Luo, X. Zhang, J. Zhang: Improving Local Search for Structured SAT Formulas via Unit Propagation Based Construct and Cut Initialization (Short Paper). CP 2021: 5:1-5:10

A Quest on efficient hybrid solvers

Cooperation Between UP-Construction and SLS

3. From UP-based Decimation to SLS and Back



S. Cai, C. Luo, H. Zhang: From Decimation to Local Search and Back: A New Approach to MaxSAT, ***IJCAI***, 571-577 (2017).

A Quest on efficient hybrid solvers

Cooperation Between UP-Construction and SLS

Unweighted Max-SAT - Industrial

Solver	#Ins.	CnC-LS	dsat-wpm3-in-ms	WPM3-2015-in
ial/circuit-debugging-problems	3	2.76(2)	10.50(3)	3.68(3)
sean-safarpour	52	51.05 (45)	43.30(37)	24.18(35)
Total	55	47	40	38

Table: Experimental Results on the MSE2016 PMS benchmarks.

Benchmark	#inst.	*DeciDist*		*Dist*	
		#win.	time	#win.	time
MSE2016_PMS_Industrial	601	**398**	84.91	225	57.31

Table: Experimental Results on MSE2016 WPMS benchmarks.

Benchmark	#inst.	*DeciCCEHC*		*CCEHC*	
		#win.	time	#win.	time
MSE2016_WPMS_Industrial	630	**319**	117.67	140	103.74

S. Cai, C. Luo, H. Zhang: From Decimation to Local Search and Back: A New Approach to MaxSAT, ***IJCAI***, 571-577 (2017).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

CDCL searches in the space of partial assignments

-->Better to integrate reasoning techniques

SLS walks in the whole search space of all complete assignments

-->Better at sampling

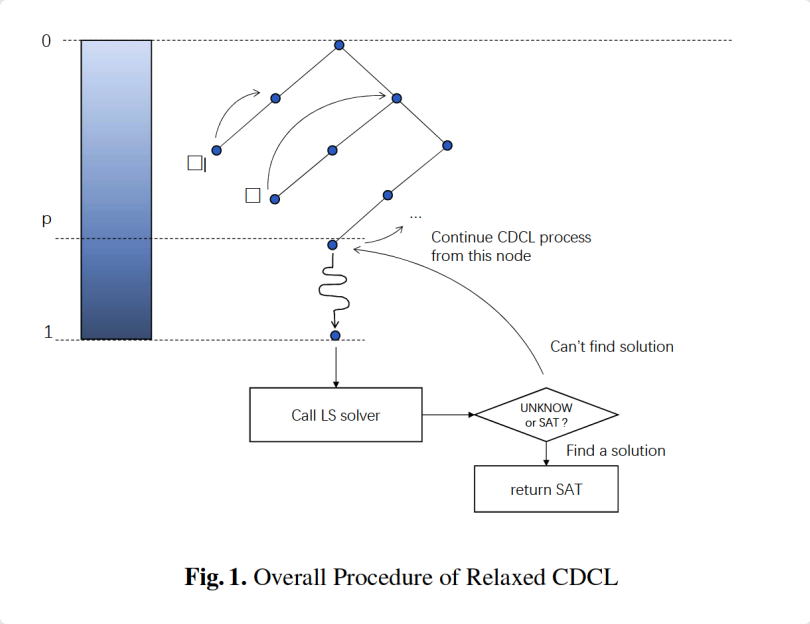


Fig. 1. Overall Procedure of Relaxed CDCL

SLS sampling ←→ CDCL solving
Boosting CDCL with SLS information

Plug SLS into a CDCL solver

- Calling SLS on promising branches
- Filter similar branches

S. Cai, X. Zhang: Deep Cooperation of CDCL and Local Search for SAT, ***SAT*** 2021 (best paper).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

(1) Exploring Promising Branches by Local Search

Identify which branches deserve exploration

$\frac{|\alpha|}{|V|} > p$ and there is no conflict under α. $p = 0.4$

$\frac{|\alpha|}{|\alpha\_max|} > q$ and there is no conflict under α. $q = 0.6$

- Explore the branch by firstly extend it to a complete assignment, and then call SLS search nearby.
- The cutoff of each Local Search process: certain amount of memory accesses (5×10^7)

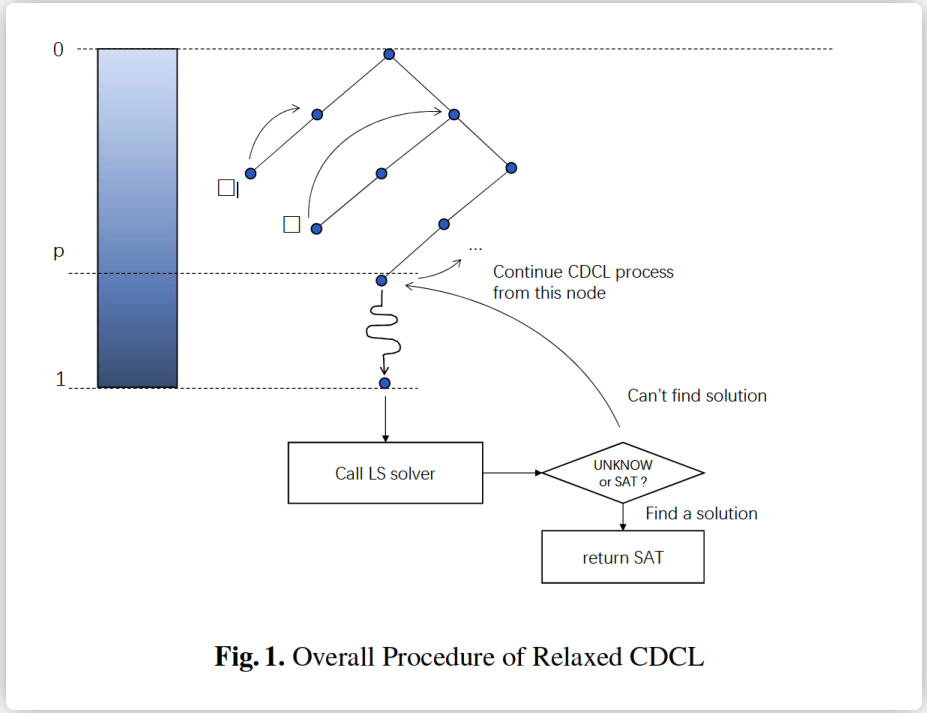


Fig. 1. Overall Procedure of Relaxed CDCL

S. Cai, X. Zhang: Deep Cooperation of CDCL and Local Search for SAT, ***SAT*** 2021 (best paper).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

(2) Rephasing with Local Search Assignments

note that rephasing has been used in Kissat [biere 2019]

Resets the saved phases of all variables with assignments produced by local search.

- After each time the CDCL is restarted [NewTechRelaxed in SAT2020]
- Fixed frequency [lstechMaple in SAT 2021]

Table 1. Probability of different phases in our phase resetting mechanism

Phase Name	$\alpha\_max\_LS[x]$	$\alpha\_latest\_LS[x]$	$\alpha\_best\_LS[x]$	no change
Probability	20%	65%	5%	10%

α_max_LS and α_best_LS serve for the aim to maximize the depth of the branch
α_latest_LS adds diversification

S. Cai, X. Zhang: Deep Cooperation of CDCL and Local Search for SAT, ***SAT*** 2021 (best paper).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

(3) Improve Branching with Conflict Frequency in Local Search

This idea has been used in NewTechRelaxed and CryptoMiniSAT-CCAnr [SoosCaiDevriendt, Gocht, Shaw, Meel] in SC2020

- CDCL is a powerful framework owing largely to the utilization of the conflict information branching heuristics aim to promote conflicts.
- Can information from SLS be used to enhance branching heuristics?

ls_confl_freq (x) = #(steps in which x appears in unsatisfied clauses) / #total_local_search_steps multiply ls_confl_freq(x) with 100 , resulting ls_confl_num(x).

LS Enhanced VSIDS: for each variable x, its activity is increased by $ls\_confl\_num(x)$

LS Enhanced LRB: for each variable x, the number of learnt clause during its period I is creased by $ls\_confl\_num(x)$.

S. Cai, X. Zhang: Deep Cooperation of CDCL and Local Search for SAT, ***SAT*** 2021 (best paper).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

- The hybrid method improves 3 typical CDCL solvers on benchmarks from SAT Competitions 2017-2020.

solver	#SAT	#UNSAT	#Solved	PAR2	#SAT	#UNSAT	#Solved	PAR2
	SC2017(351)				SC2018(400)			
glucose_4.2.1	83	101	184	5220.0	95	95	190	5745.9
glucose+rx	88	95	183	5220.0	113	95	208	5745.9
glucose+rx+rp	112	94	206	4618.2	141	87	228	4698.3
glucose+rx+rp+cf	110	94	204	4668.5	150	91	241	4438.2
Maple-DL-v2.1	101	113	214	4531.0	133	102	235	4533.9
Maple-DL+rx	101	112	213	4592.6	149	101	250	4148.4
Maple-DL+rx+rp	111	103	214	4447.1	158	93	251	4404.2
Maple-DL+rx+rp+cf	116	107	223	4139.4	162	97	259	3927.6
Kissat_sat	115	114	229	3943.5	167	98	265	3786.4
Kissat_sat+cf	113	113	226	4060.0	178	104	282	3400.4
CCAnr	13	N/A	13	9629.9	56	N/A	56	8622.0
	SC2019(400)				SC2020(400)			
glucose_4.2.1	118	86	204	5437.6	68	91	159	6494.6
glucose+rx	120	84	204	5412.0	93	88	181	6012.4
glucose+rx+rp	134	85	219	5230.3	130	85	215	5122.7
glucose+rx+rp+cf	140	85	225	4923.6	134	87	221	4977.9
Maple-DL-v2.1	143	97	240	4601.8	86	104	190	5835.7
Maple-DL+rx	146	93	239	4602.4	121	105	226	4877.8
Maple-DL+rx+rp	155	94	249	4413.3	142	99	241	4389.2
Maple-DL+rx+rp+cf	154	95	249	4377.4	151	106	257	4171.1
Kissat_sat	159	88	247	4293.5	146	114	260	4403.8
Kissat_sat+cf	162	90	252	4211.7	157	113	270	3896.8
CCAnr	13	N/A	13	9678.3	45	N/A	45	8978.7

	Analysis for SAT				Analysis for UNSAT	
solver	#byLS	#SAT_bonus	#LS_call	LS_time(%)	#LS_call	LS_time(%)
	SC2017(351)					
glucose+rx	20	11	24.28	21.66	16.36	5.52
glucose+rx+rp	10	33	17.77	18.46	14.33	4.86
glucose+rx+rp+cf	17	29	22.7	22.19	15.3	5.81
Maple+rx	16	9	13.86	7.52	11.18	2.03
Maple+rx+rp	11	15	9.63	10.43	6.54	2.36
Maple+rx+rp+cf	6	16	12.59	7.49	8.59	2.12
	SC2018(400)					
glucose+rx	50	4	11.27	20.66	29.62	4.94
glucose+rx+rp	47	31	9.46	18.4	21.66	5.64
glucose+rx+rp+cf	53	36	11.43	20.28	20.62	6.64
Maple+rx	52	7	4.8	13.02	11.69	2.81
Maple+rx+rp	56	13	4.84	15.21	8.7	3.04
Maple+rx+rp+cf	51	18	6.52	12.53	15.62	2.94
	SC2019(400)					
glucose+rx	14	8	26.46	10.79	17.42	6.39
glucose+rx+rp	10	26	22.68	8.67	14.59	5.14
glucose+rx+rp+cf	11	26	20.39	11.82	15.51	5.95
Maple+rx	14	7	12.66	2.67	12.94	1.98
Maple+rx+rp	9	14	8.6	3.17	16.59	2.53
Maple+rx+rp+cf	12	15	11.21	3.05	17.23	2.22
	SC2020(400)					
glucose+rx	30	9	14.94	11.75	14.67	10.27
glucose+rx+rp	23	37	13.17	10.79	9.4	9.71
glucose+rx+rp+cf	23	37	12.78	11.67	10.52	10.28
Maple+rx	19	13	14.21	6.69	10.24	5.25
Maple+rx+rp	30	29	8.53	6.62	11.7	6.18
Maple+rx+rp+cf	23	36	10.95	6.05	14.17	5.42

S. Cai, X. Zhang: Deep Cooperation of CDCL and Local Search for SAT, ***SAT*** 2021 (best paper).

A Quest on efficient hybrid solvers

Deep cooperation between CDCL Solving + SLS Sampling

- The hybrid solvers in our team achieved good results in recent SAT/MaxSAT/SMT competitions.

SAT Competitions

- Main track SAT 1st , 2020
- Incremental track 1st , 2020
- Planning track 2nd , 2020
- Main track SAT/UNSAT 2nd 2021

SMT Competition 2021

QF_IDL 1st, 2021

MaxSAT Evaluation 2021

Complete track: unweighted 1st , weighted 2nd

Incomplete track: unweighted 1st , weighted 1st

The ongoing trend:
Most (if not all) top-3 winners of SAT Competition 2021 are based on hybrid solvers.

Thank you!

Welcome to visit us in Beijing! ☺

Email: caisw@ios.ac.cn