“约束求解基础与应用”训练营 2022

A Tutorial to SAT Solving

Shaowei Cai

Institute of Software, Chinese Academy of Sciences

2022.4.10

课程信息

参考书 Handbook of Satisfiability

关于作业和结业证书

- 部分课程内容会布置作业，分为不同难度，作业在结营之前提交。
- 标注“必做题目”的作业至少完成一道，完成即可获得结业证书。
- 根据作业的完成情况评出优秀学员，颁发优秀学员证书和奖励。
- 助教：
  - 张昕荻 dezhangxd@163.com
  - 李博涵 libohan19@mails.ucas.ac.cn
  - 何锦龙 hejl@ios.ac.cn

Constraint Solving 约束求解

$$\begin{tikzcd}\text{Problem} \arrow[r] & \boxed{\text{Encoding}} \arrow[r] & \text{Mathematical sentence } \Phi \arrow[r] & \boxed{\text{Solving}} \arrow[r] & \text{Solution}\end{tikzcd}$$

Complete Solvers

Incomplete Solvers

SAT:

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

MP:

$$y^2 + 2xy < 100$$
$$x \leq 60$$

谓词逻辑（SMT）：

$$((y^2 + 2xy < 100) \vee ((f(y) < 30) \rightarrow \neg(x - y < 30) \wedge (x \leq 60))$$

CSP:

$$AllDifferent(x_i, x_j) \wedge |x_i - x_j| \neq |i - j|$$

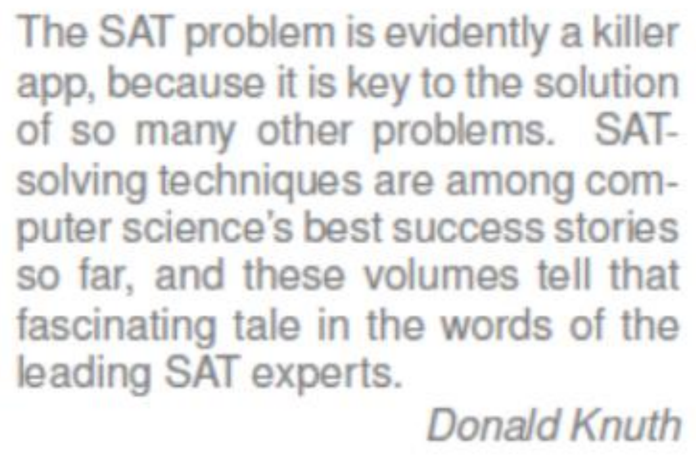
Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search

Hello, SAT!

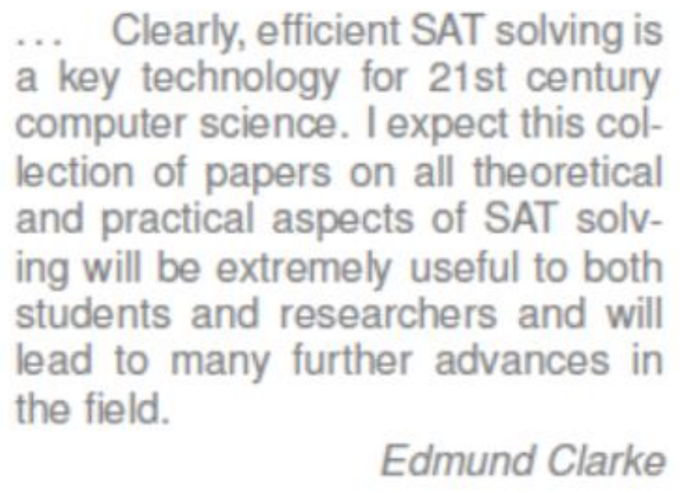
- Propositional Satisfiability, aka. Boolean Satisfiability, abbreviated as SAT

- The first NP-Complete problem [Cook, 1971]
- A core problem in computer science and a basic problem in logic
- SAT solvers widely used in industry and science



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

SAT

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

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

- 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

  $x_2 \vee x_3$,

  $\neg x_1 \vee \neg x_3 \vee x_4 \quad (\{\neg x_1, \neg x_3, x_4\})$

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

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.

SAT

Example: Determine the satisfiability of the following compound propositions:

$$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

Solution: Satisfiable. Assign **T** to *p, q,* and *r.*

$$(p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p) \wedge (p \vee q \vee r) \wedge (\neg p \vee \neg q \vee \neg r)$$

Solution: Not satisfiable. Check each possible assignment of truth values to the propositional variables and none will make the proposition true.

Dichotomy theorem of SAT

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

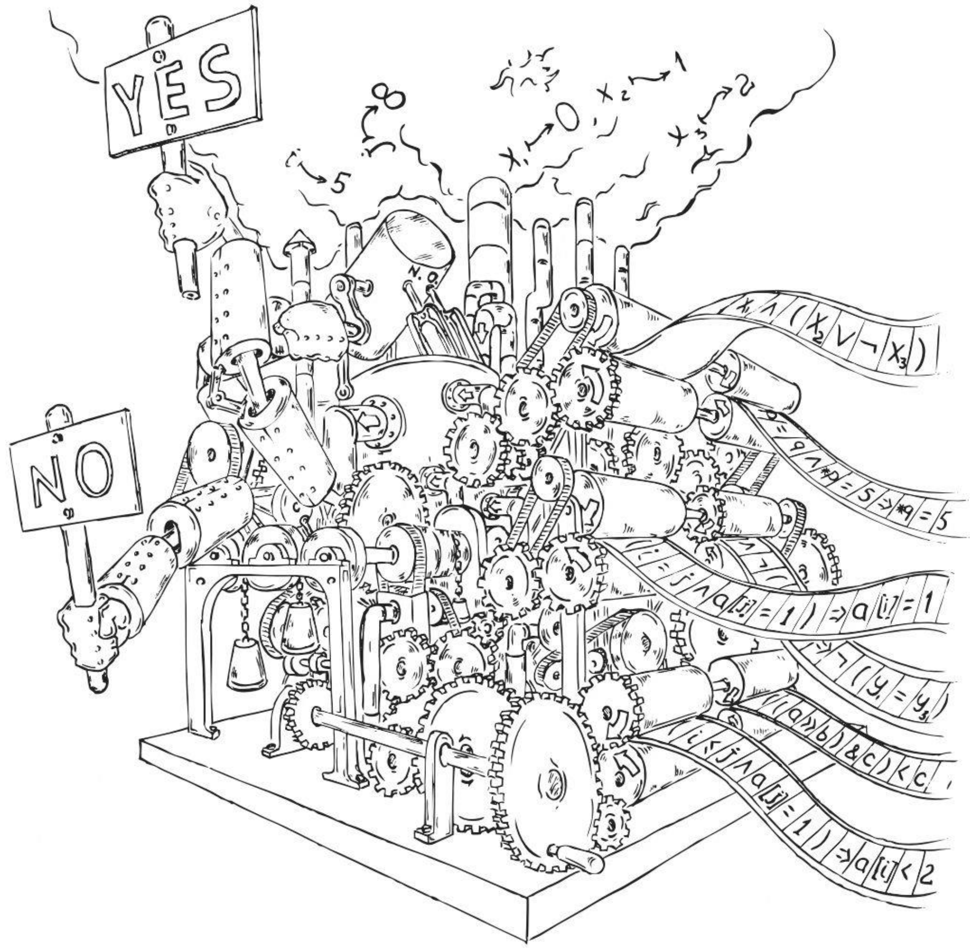
Theorem (Schaefer, STOC 1978)

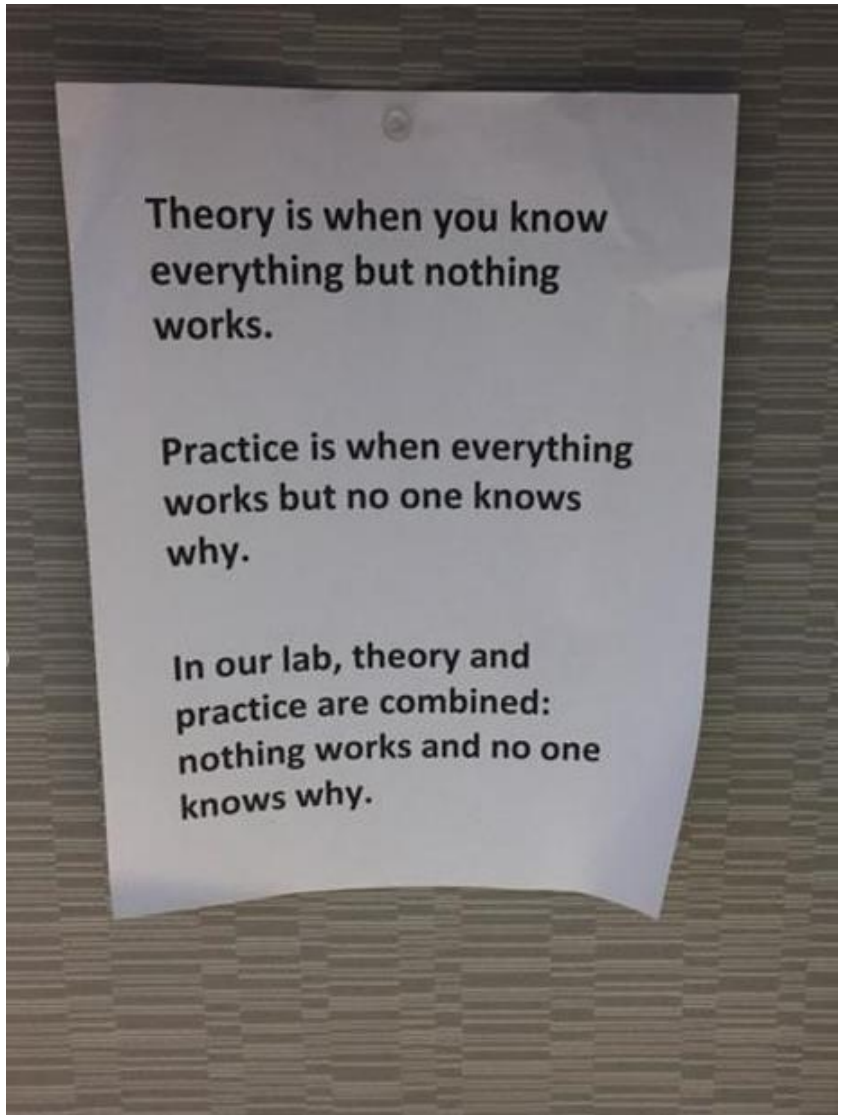
Given a Boolean constraint set C, SAT(C) is in P if C satisfies one condition below, and otherwise it is NP-complete:

- 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

Solvers 求解器





Theory is when you know everything but nothing works.

Practice is when everything works but no one knows why.

In our lab, theory and practice are combined: nothing works and no one knows why.

SAT solvers

- Using SAT solvers
  - To find a certain structure
  - To prove something

Constraints
Scheduling,
Resource allocation,
Logistics,
Hardware design,
Software Engineering,
…

Reasoning
Theorem proving,
System verification,
Knowledge representation,
Decision procedure,
Agents,
…

SAT ty

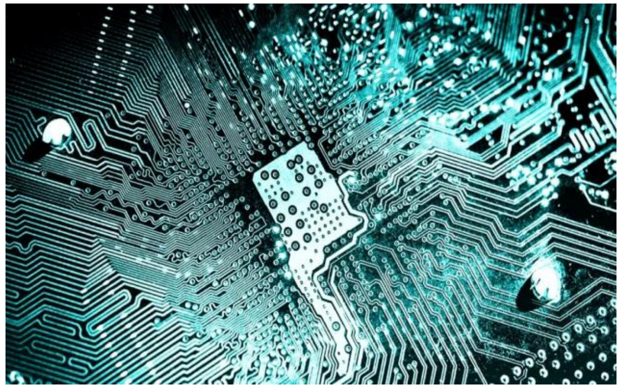
Finding a solution

Prove that
$x \geq 1, y \geq 1 \Rightarrow x + y \geq 2$

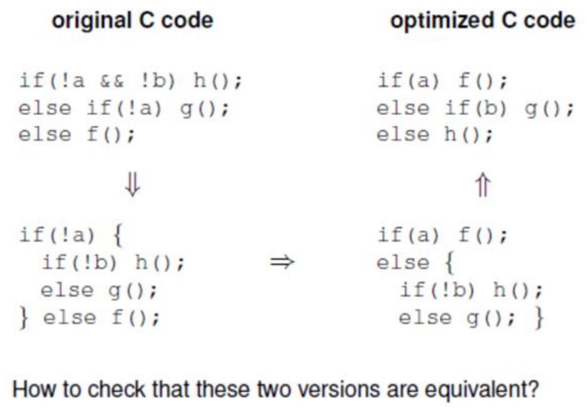
Is
$x \geq 1, y \geq 1, not(x + y \geq 2)$
UNSAT?

Checking validity

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

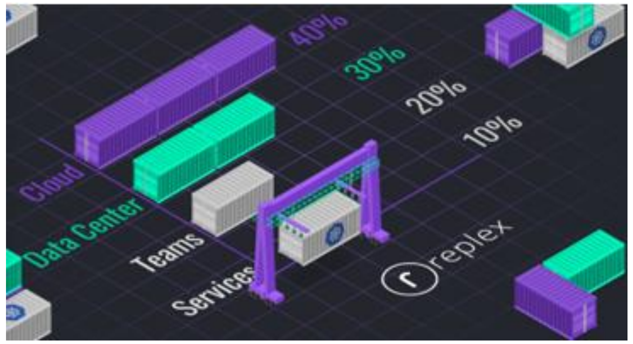
$x^m + y^m = z^m \pmod{p}$ *$vdW(6) = 1132$*

Schur's Theorem *Ramsey Theory*

Pythagorean Tuples Conjecture

3n+1 Conjecture?

Math



Resource Allocation

Using SAT Solvers

MiniSat: A open-source SAT solver widely used in industries.

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
```

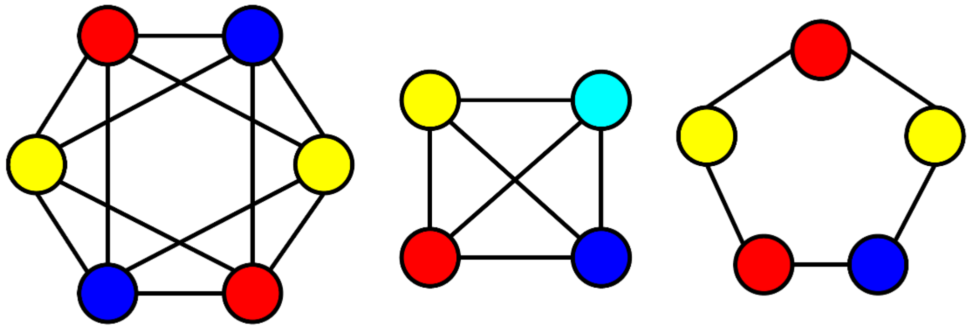
the solution line (starting with ”s”) can contain SATISFIABLE, UNSATISFIABLE and UNKNOWN.
For SATISFIABLE case, the truth values of variables are printed in lines starting with ”v”, the last value is followed by a ”0”

Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search

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.

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

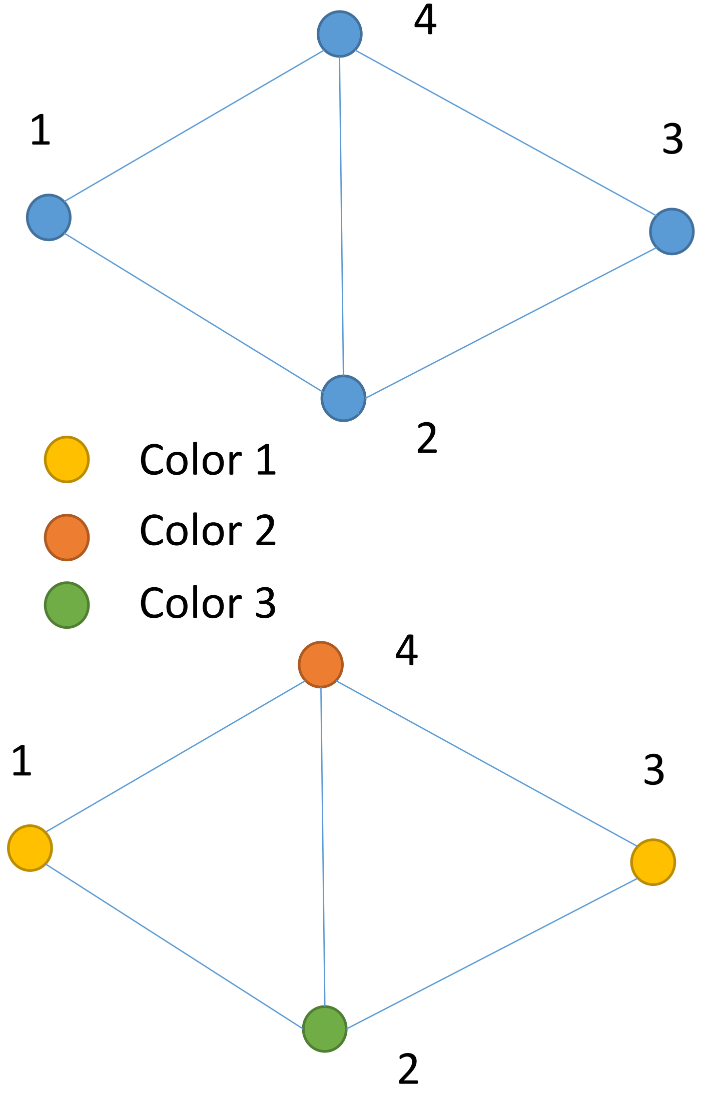
  - *For each edge, produces 3 negative 2-clause*

    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:

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



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

SAT Encoding: Pythagorean Tuples Conjecture

Problem Definition:

Is it possible to assign to each integer 1,2,.....n one of two colors such that if $a^2 + b^2 = c^2$ then a, b and c do not all have the same color?

- Solution : Nope
- for n=7825 it is not possible
- proof obtained by a SAT solver (2016)

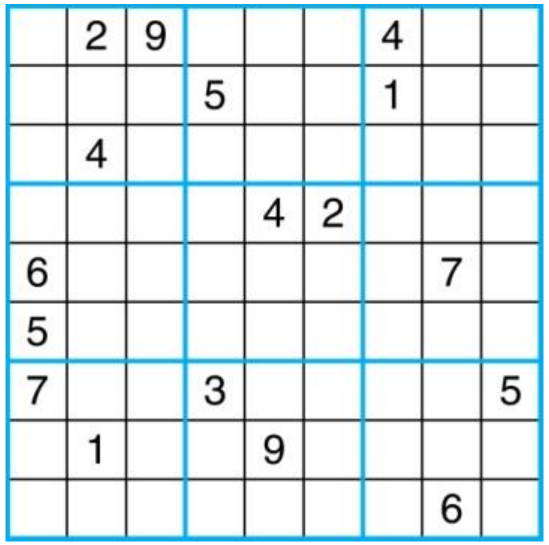
How to encode this?

- for each integer i we have a Boolean variable x_i, x_i= 1 if color of i is 1, x_i=0 otherwise.
- for each a, b, c such that $a^2 + b^2 = c^2$ we have two clauses: $(x_a \lor x_b \lor x_c)$ and $(\bar{x}_a \lor \bar{x}_b \lor \bar{x}_c)$

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



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

Encoding Sudoku to SAT

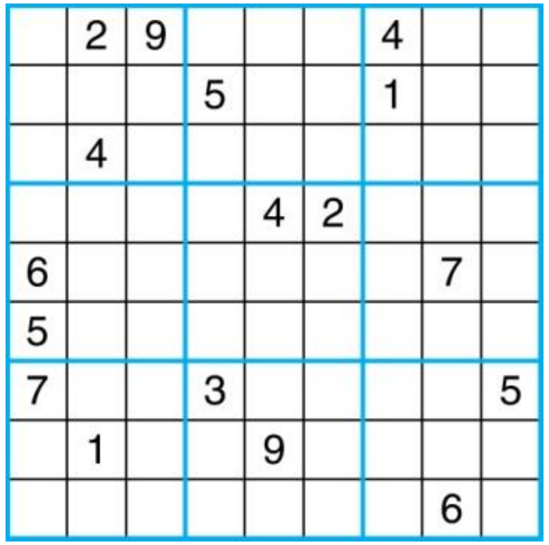
- For each cell with a given value, assert *p*(*i*,*j*,*n*), when the cell in row *i* and column *j* has the given value.

- 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.

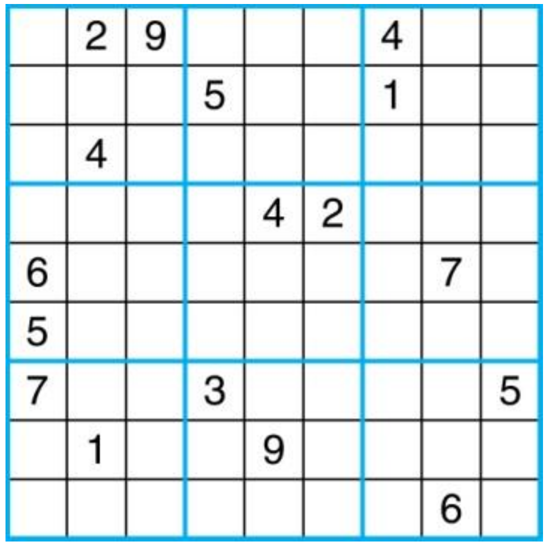
$$\bigwedge_{r=0}^{2}\bigwedge_{s=0}^{2}\bigwedge_{n=1}^{9}\bigvee_{i=1}^{3}\bigvee_{j=1}^{3} p(3r+i, 3s+j, n)$$

- Assert that no cell contains more than one number.

$$n \neq n'$$

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

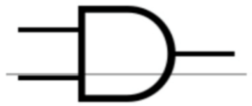
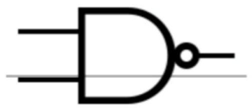
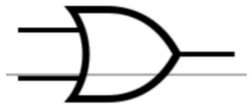
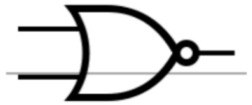
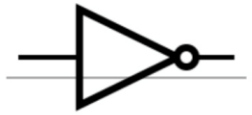
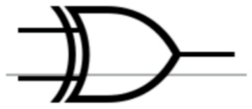
通过枚举二元负文字对保证独一性



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

Encoding Circuit to CNF

Tseitin Transformation

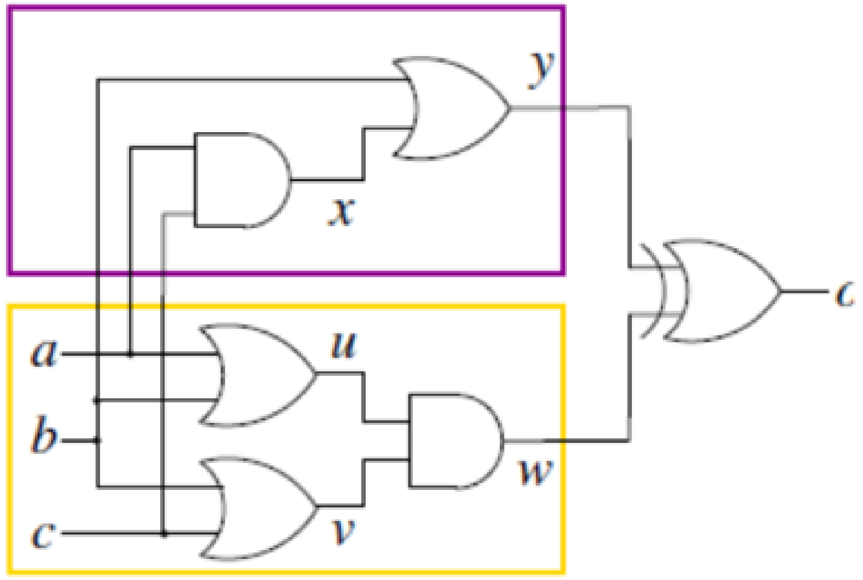
Type	Operation	CNF Sub-expression
 AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
 NAND	$C = \overline{A \cdot B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
 OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
 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	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$

$$C = A \cdot B$$

$$C \to A \wedge B \equiv \neg C \vee (A \wedge B)$$
$$\equiv (A \vee \neg C) \wedge (B \vee \neg C)$$

$$A \wedge B \to C \equiv \neg(A \wedge B) \vee C$$
$$\equiv \neg A \vee \neg B \vee C$$

Encoding Circuit to CNF



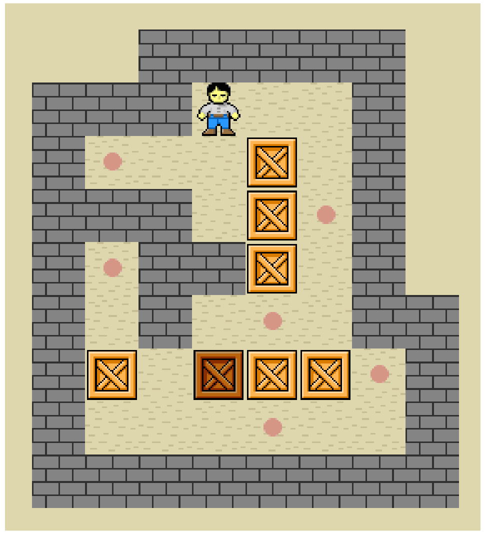
$$
\begin{aligned}
& o\ \wedge \\
(x &\leftrightarrow a \wedge c) \wedge \\
(y &\leftrightarrow b \vee x) \wedge \\
(u &\leftrightarrow a \vee b) \wedge \\
(v &\leftrightarrow b \vee c) \wedge \\
(w &\leftrightarrow u \wedge v) \wedge \\
(o &\leftrightarrow y \oplus w)
\end{aligned}
$$

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

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

Encoding Sokoban 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]

Planning Problem Definition

- 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 $\left(pre(a), eff(a)\right)$

  $pre(a)$ 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 $eff(a)$)

Planning Problem Definition

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 will 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

Planning Problem Definition

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

Planning Problem Definition

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:

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

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

Planning Problem Definition

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:

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

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

Planning Problem Definition

If an action is applied it must be applicable:

(preconditions是action的必要条件 $a^t \rightarrow \wedge_{\forall(x=v)\in pre(a)} b^t_{x=v}$)

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

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

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

(action是effects的充分条件 $a^t \rightarrow \wedge_{\forall(x=v)\in eff(a)} b^{t+1}_{x=v}$)

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

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

Planning Problem Definition

State variables cannot change without an action between steps

$$(\neg b^t_{x=v} \wedge b^{t+1}_{x=v}) \rightarrow a^t_{s_1} \vee \cdots \vee a^t_{s_j}$$
$$\Leftrightarrow (b^t_{x=v} \vee \neg b^{t+1}_{x=v} \vee a^t_{s_1} \vee \cdots \vee a^t_{s_j})$$

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

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.

对于某个位置x，如果第t步它不在状态x=v,而第t+1步它在状态x=v,
则必定是发生了某个支持x=v的action

Planning Problem Definition

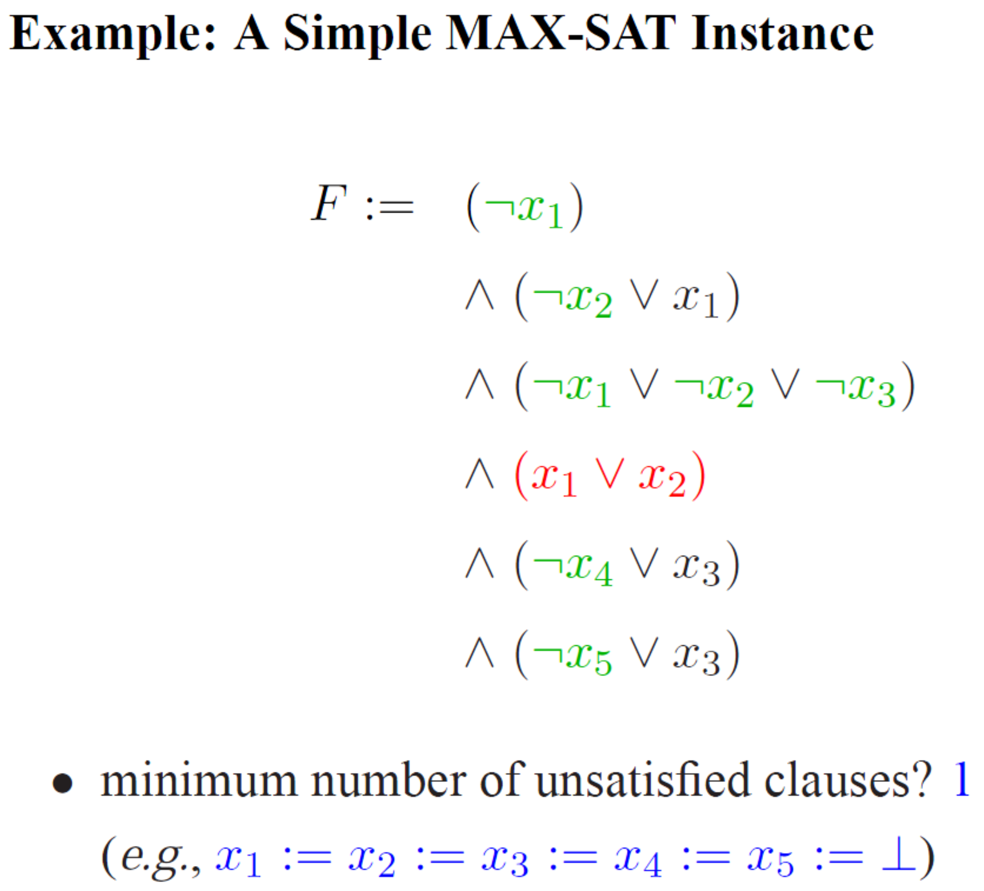
At most one action is used in each step:

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

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

MaxSAT

- When the formula is not satisfiable, we concern about satisfying as many clauses as possible -> Maximum Satisfiability.



Example: A Simple MAX-SAT Instance

$$\begin{aligned} F := \quad & (\neg x_1) \\ & \wedge (\neg x_2 \vee x_1) \\ & \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\ & \wedge (x_1 \vee x_2) \\ & \wedge (\neg x_4 \vee x_3) \\ & \wedge (\neg x_5 \vee x_3) \end{aligned}$$

- minimum number of unsatisfied clauses? 1

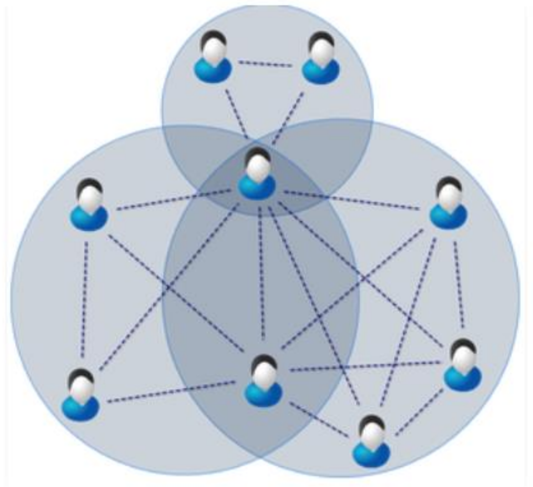
  (*e.g.*, $x_1 := x_2 := x_3 := x_4 := x_5 := \bot$)

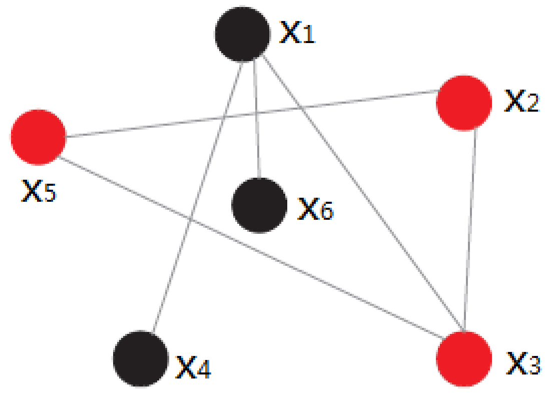
Variants of MaxSAT

- Weighted MaxSAT
  - Each clause is associated with a weight, the goal: maximize the total weight of satisfied clauses
- Partial MaxSAT
  - hard clauses: must be satisfied
  - soft clauses: to satisfy as many as possible
  - the goal: satisfy all hard clauses and as many soft clauses as possible.
- Weighted Partial MaxSAT
  - Each soft clause is associated with a weight
  - The goal: satisfy all hard clauses and maximize the total weight of satisfied soft clauses.

Encoding MaxClique to MaxSAT

- MaxClique Problem





A **clique** is a vertex subset C such that every vertex in C is adjacent to any other vertices in C.

- hard clauses:

$\neg x_1 \vee \neg x_5$

$\neg x_1 \vee \neg x_2$

$\dots$

$\neg x_i \vee \neg x_j, \forall e_{ij} \notin E$

- soft clauses:

x_1

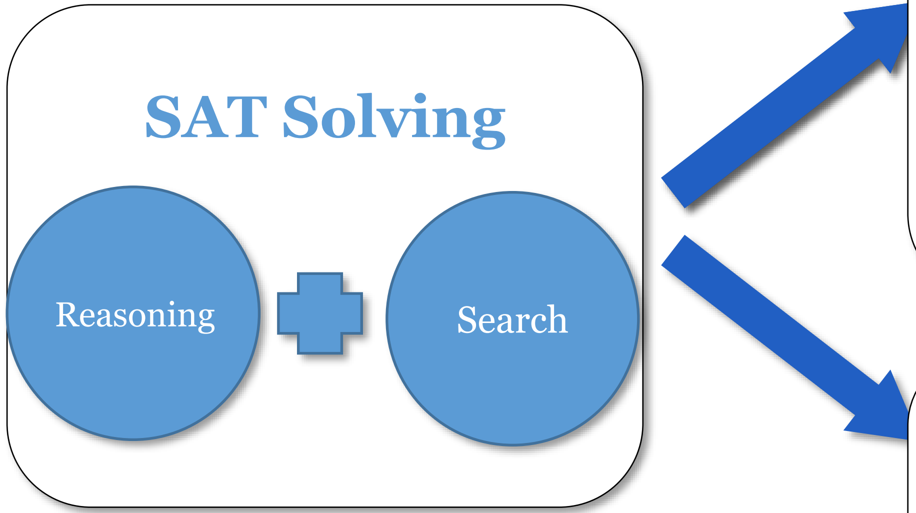
$\dots$

x_6

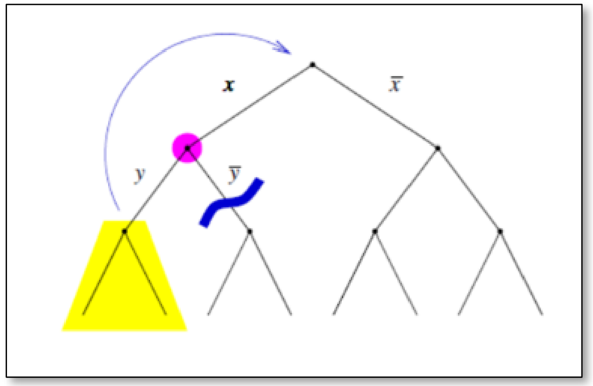
Outline

- SAT Basis
- SAT Encoding
- CDCL
- Local Search

SAT Solving Basis

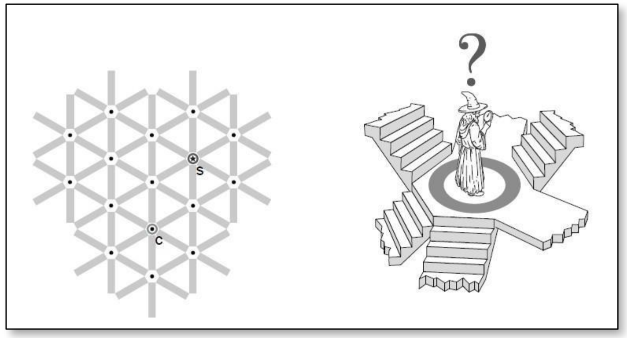


Complete Solvers： conflict-driven clause learning



CDCL

Incomplete Solvers： biased on satisfiable side



Stochastic
local search

SAT Solving Basis – Resolution 归结

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

- **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 $A \cup B \setminus \{a, \neg a\}$ is called the resolvent of A and B (by a) and denoted by $R(A, B)$.

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. $F = (x \lor e) \land (y \lor e) \land (\overline{x} \lor z \lor \overline{e}) \land (y \lor \overline{e}) \land (y \lor z)$

Eliminating variable e by resolution:

- first add all resolvents upon e.

  $\{(x \lor e), (y \lor e)\}$ with $\{(x \lor z \lor \overline{e}), (y \lor \overline{e})\} \rightarrow$ 4 resolvents

  $$F \land (x \lor \overline{x} \lor z) \land (x \lor y) \land (y \lor \overline{x} \lor z) \land (y)$$

- remove all clauses that contain e to obtain

$$(y \lor z) \land (x \lor \overline{x} \lor z) \land (x \lor y)\ (y \lor \overline{x} \lor z) \land (y)$$

SAT Solving Basis - Unit Propagation 单元传播

- 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			$\overline{x_1} \vee \overline{x_2}$	
$Vars:$	x_3		$clauses:$	$\overline{x_1} \vee \overline{x_3}$	
	x_4			$x_2 \vee \overline{x_3} \vee \overline{x_4}$	
	x_5			$\overline{x_1} \vee x_4 \vee x_5$	
	x_6			$x_2 \vee x_4 \vee x_6$	

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:

$$\begin{array}{llllll}
 & x_1 & T & & x_1 & sat \\
 & x_2 & & & \cancel{\bar{x}_1} \vee \bar{x}_2 & \\
Vars: & x_3 & & clauses: & \cancel{\bar{x}_1} \vee \bar{x}_3 & \\
 & x_4 & & & x_2 \vee \bar{x}_3 \vee \bar{x}_4 & \\
 & x_5 & & & \cancel{\bar{x}_1} \vee x_4 \vee x_5 & \\
 & x_6 & & & x_2 \vee x_4 \vee x_6 &
\end{array}$$

SAT Solving Basis – DP Algorithm

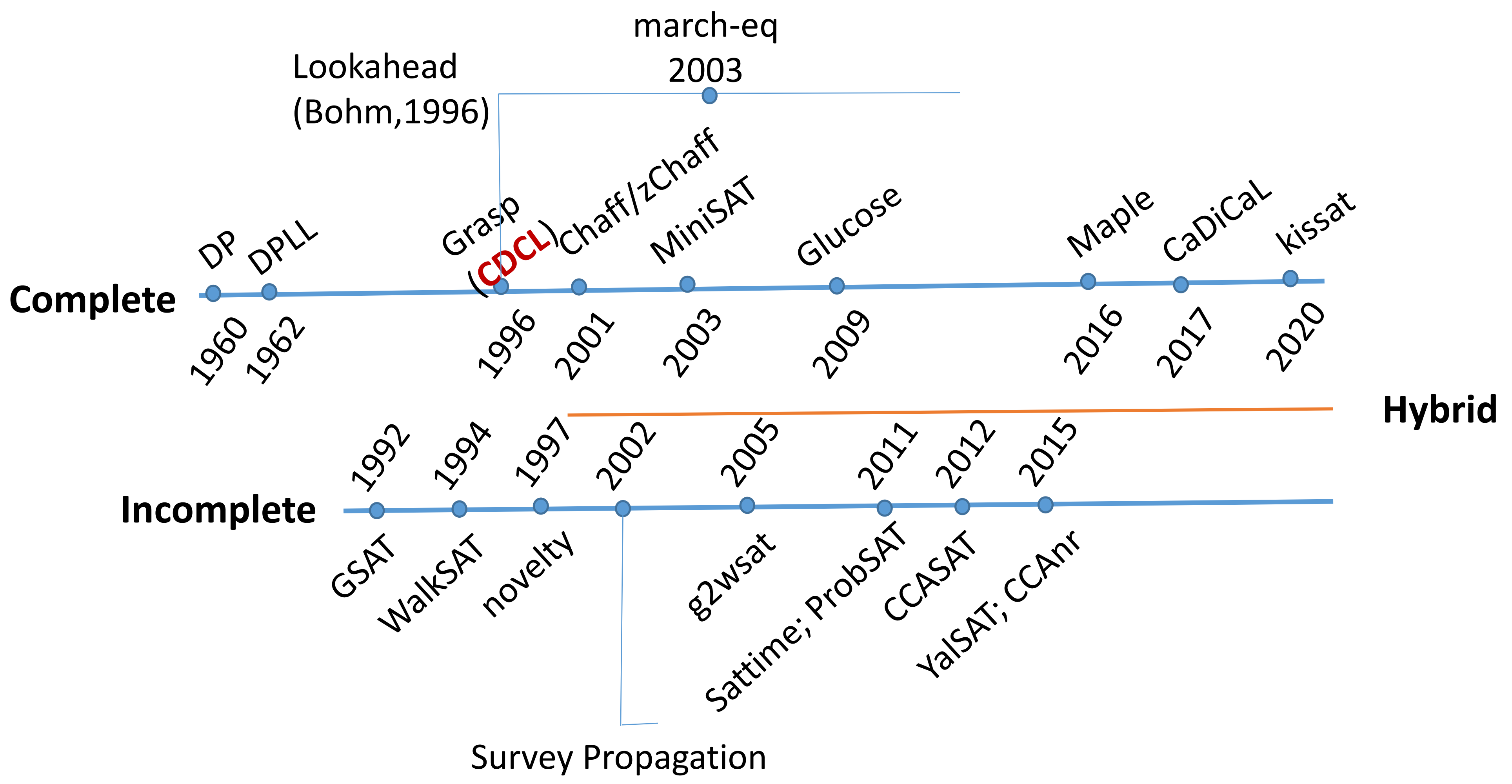
Davis-Putnam Algorithm [1960] % could find record in 1959

- 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



A brief history of SAT solving

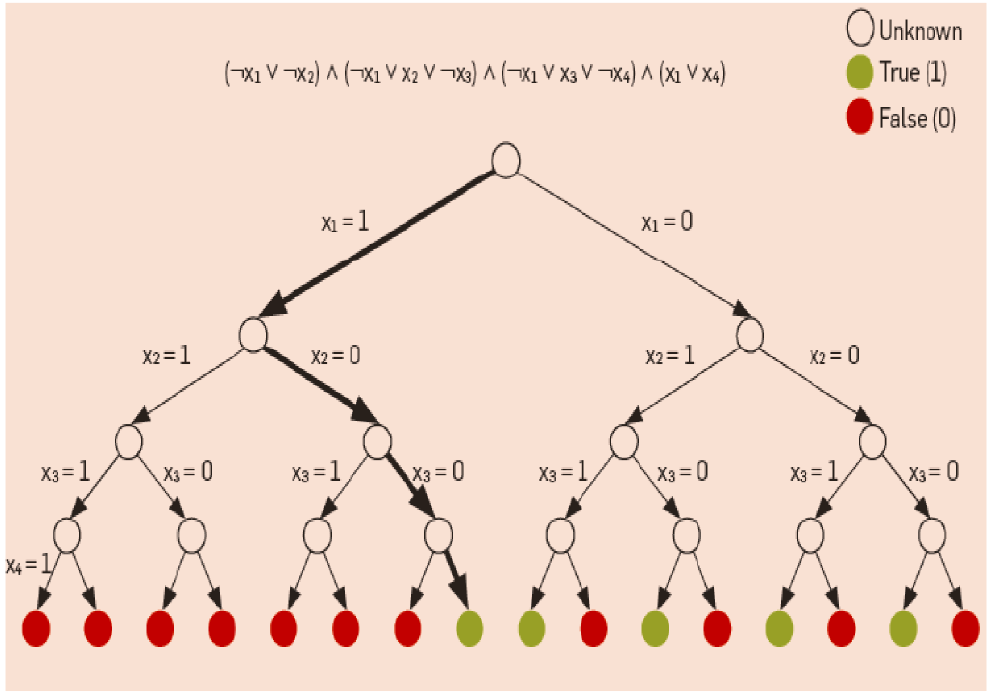
- 1960-1990
  - DP, DPLL
  - NP completeness, Complexity, Tractable subclass,…
  - Resolution: Stålmarck's Method (1989)
- 1990-2010
  - Local Search (1992): GSAT (1992), WalkSAT (1994)
  - Conflict Driven Clause Learning (1996): GRASP(1999), Chaff(2000), MiniSAT (2003), Glucose (2009)
  - Phase transition, Survey Propagation
  - Portfolio: SATzilla (2007)
- 2010~today
  - Modern local search
  - Advanced clause management and simplification
  - Hybridizing CDCL and local search (winners in 2020 and 2021 are of this type)
  - Parallel solving: cube and conquer

Millions of variables solved in 1 hour regularly

DPLL Algorithm

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

• Chronological backtracking + UP + Decision heuristics


$(\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_4)$
Unknown
True (1)
False (0)
$x_1 = 1$
$x_1 = 0$
$x_2 = 1$
$x_2 = 0$
$x_3 = 1$
$x_3 = 0$
$x_4 = 1$

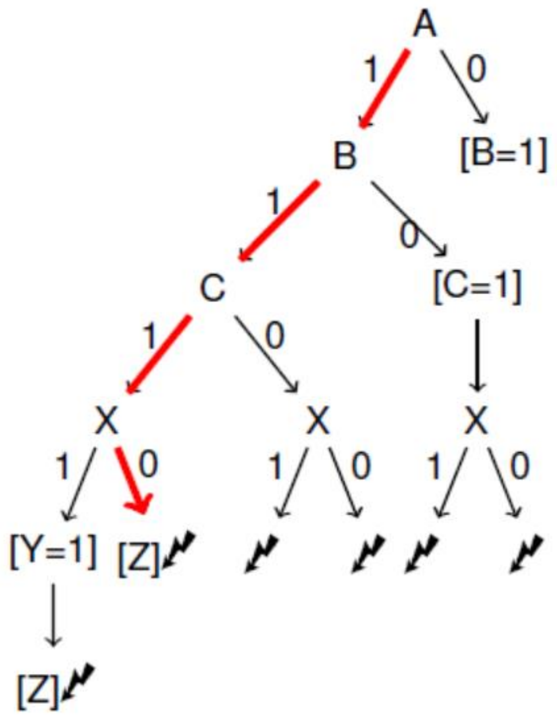
conditioning Δ on literal L: $\Delta|L = \{\alpha - \{\neg L\} \mid \alpha \in \Delta,\ L \notin \alpha\}.$

DPLL Algorithm

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

- Chronological backtracking + UP + Decision heuristics

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



Decision level

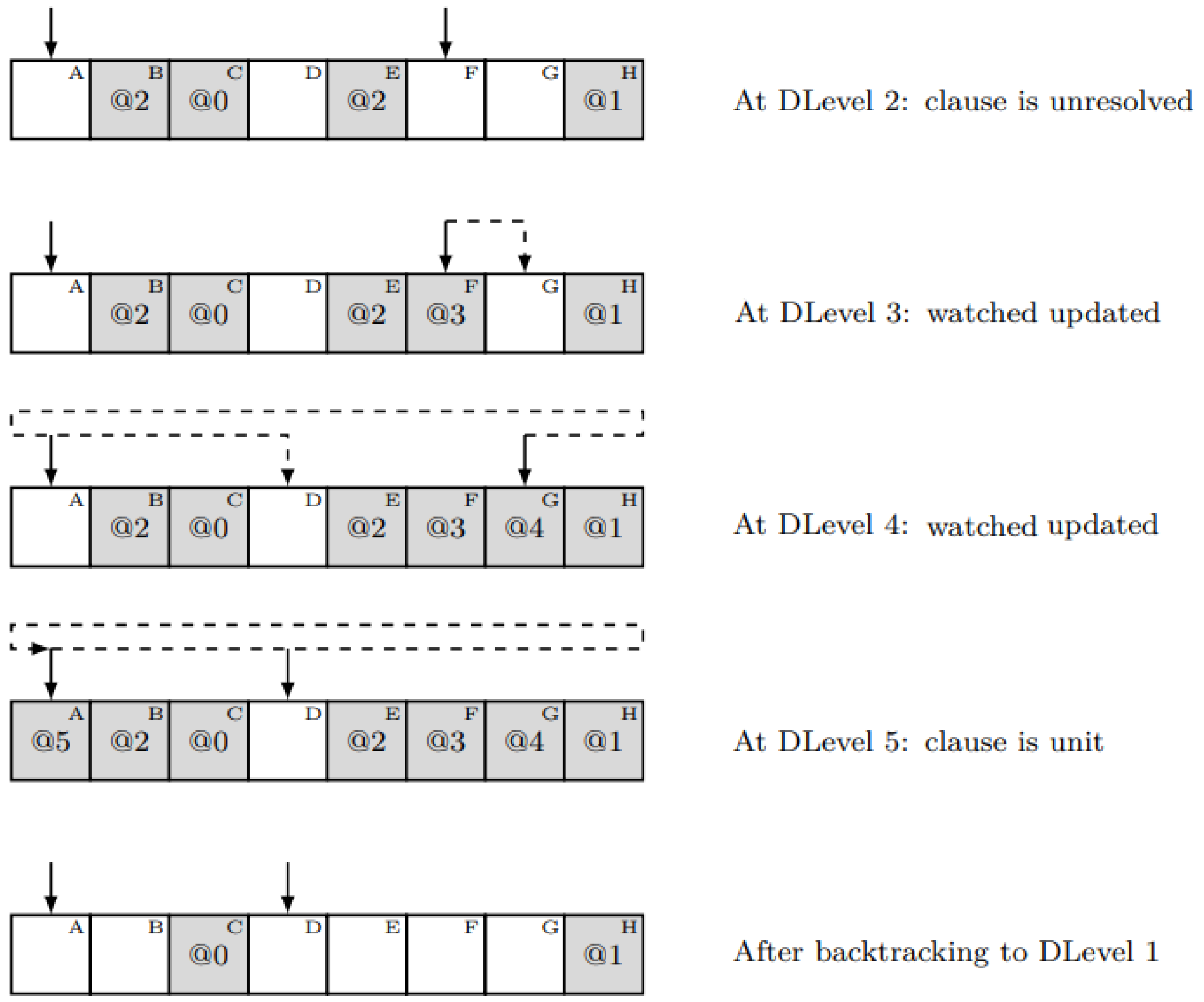
$\underbrace{\text{[UP]}}_{\text{Level 0}}\ \underbrace{\text{Decide [UP]}}_{\text{Level 1}}\ \text{Decide [UP] ....}$

Lazy data structure for UP

[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


A
B
C
D
E
F
G
H
@2
@0
@2
@1
At DLevel 2: clause is unresolved
@3
At DLevel 3: watched updated
@4
At DLevel 4: watched updated
@5
At DLevel 5: clause is unit
After backtracking to DLevel 1

VSIDS Branching heuristics

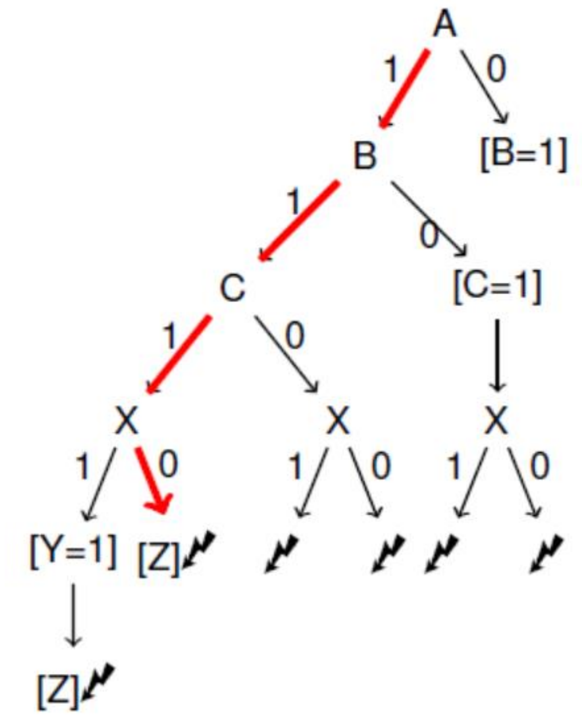
- Branching heuristics are used for deciding which variable to use when branching.
  - Solvers prefer the variable which may cause conflicts faster.
- Variable State Independent Decaying Sum (VSIDS) [MoskewiczMadiganZhaoZhangMalik'01]
  - **Compute score for each variable, select variable with highest 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

VSIDS 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.

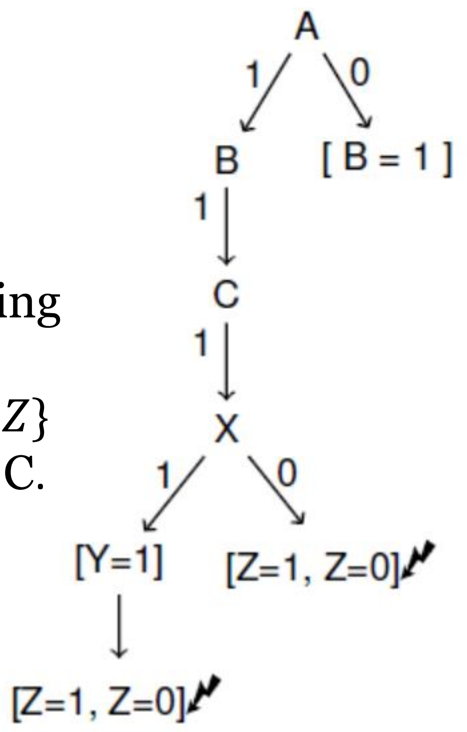
Backjumping

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



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: Conflict Driven Clause Learning

- A demonstration on clause learning

$$\varphi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg b \vee e) \wedge (\neg d \vee \neg e \vee f) \ldots$$

  - Assume decisions $c = 0$ and $f = 0$
  - Assign $a = 0$ and imply assignments
  - A conflict is reached: $(\neg d \vee \neg e \vee f)$ is unsatisfied
  - $(a = 0) \wedge (c = 0) \wedge (f = 0) \Rightarrow (\varphi = 0)$
  - $(\varphi = 1) \Rightarrow (a = 1) \vee (c = 1) \vee (f = 1)$

  - Learn new clause $(a \vee c \vee f)$

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).

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

$$\begin{tikzcd}
x_{31} = 0 @ 3 \arrow[dr, "\omega_1"] & & & & \\
& x_2 = 0 @ 5 \arrow[dr, "\omega_3"] & & x_5 = 0 @ 5 \arrow[dr, "\omega_6"] & \\
x_1 = 0 @ 5 \arrow[ur, "\omega_1"] \arrow[dr, "\omega_2"'] & & x_4 = 1 @ 5 \arrow[ur, "\omega_4"] \arrow[dr, "\omega_5"'] & & \kappa \\
& x_3 = 0 @ 5 \arrow[ur, "\omega_3"'] & & x_6 = 0 @ 5 \arrow[ur, "\omega_6"'] & \\
& & x_{21} = 0 @ 2 \arrow[ur, "\omega_5"'] & &
\end{tikzcd}$$

CDCL – Implication Graph

Implication Graph

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

$$\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & & & \\
0 & \emptyset & & & & \\
1 & h \arrow[rrrddd, bend left, "\mathfrak{c}_4"] & & & & \\
2 & b \arrow[rdd, bend left, "\mathfrak{c}_1"] & & & & \\
3 & y & & & & \\
4 & a \arrow[r, "\mathfrak{c}_1"] \arrow[rd, "\mathfrak{c}_2"'] & c \arrow[r, "\mathfrak{c}_3"] & e \arrow[r, "\mathfrak{c}_4"] \arrow[rd, "\mathfrak{c}_5"'] & f \arrow[r, "\mathfrak{c}_6"] & \perp \\
 & & d \arrow[ru, "\mathfrak{c}_3"'] & & g \arrow[ru, "\mathfrak{c}_6"'] &
\end{tikzcd}$$

(a) Implication graph

$x \in \mathbb{V} \cup \{\perp\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
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	$\mathfrak{c}_5$
$\perp$	–	–	$\mathfrak{c}_6$

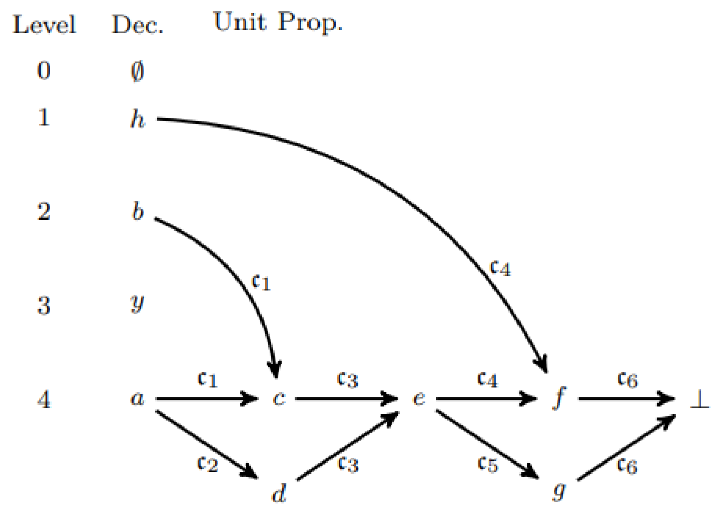
(b) State of variables

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

$x \in \mathbb{V} \cup \{\perp\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
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	$\mathfrak{c}_5$
$\perp$	–	–	$\mathfrak{c}_6$

(b) State of variables

First UIP clause learning

Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\perp$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
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	$[]$	–	–	$\{\bar{h}, \bar{e}\}$	–

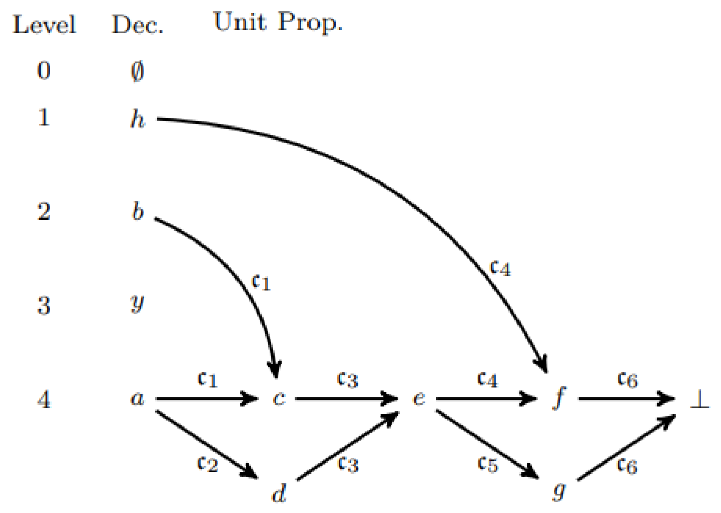
Variables are analyzed in a first-in first-out fashion, starting from the conflict.

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

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 $\perp$.

CDCL – Conflict Analysis

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



$$\begin{tikzcd}
\text{Level} & \text{Dec.} & \text{Unit Prop.} & & & \\
0 & \emptyset & & & & \\
1 & h \arrow[dddrrr, bend left, "\mathfrak{c}_4"] & & & & \\
2 & b \arrow[ddr, bend left, "\mathfrak{c}_1"] & & & & \\
3 & y & & & & \\
4 & a \arrow[r, "\mathfrak{c}_1"] \arrow[dr, "\mathfrak{c}_2"'] & c \arrow[r, "\mathfrak{c}_3"] & e \arrow[r, "\mathfrak{c}_4"] \arrow[dr, "\mathfrak{c}_5"'] & f \arrow[r, "\mathfrak{c}_6"] & \perp \\
 & & d \arrow[ur, "\mathfrak{c}_3"'] & & g \arrow[ur, "\mathfrak{c}_6"'] &
\end{tikzcd}$$

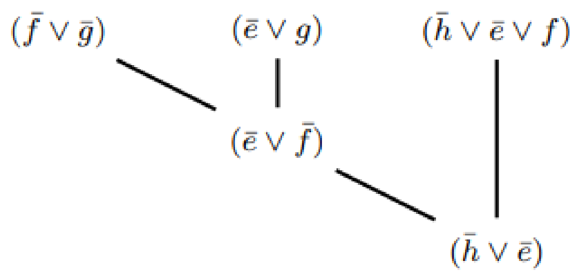
(a) Implication graph

$x \in \mathbb{V} \cup \{\perp\}$	$\nu(\cdot)$	$\delta(\cdot)$	$\alpha(\cdot)$
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	$\mathfrak{c}_5$
$\perp$	–	–	$\mathfrak{c}_6$

(b) State of variables

First UIP clause learning

Step	Var Queue	Extract Var	Antecedent	Recorded Lits	Added to Queue
0	–	$\perp$	$\mathfrak{c}_6$	$\emptyset$	$\{f, g\}$
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	$[]$	–	–	$\{\bar{h}, \bar{e}\}$	–

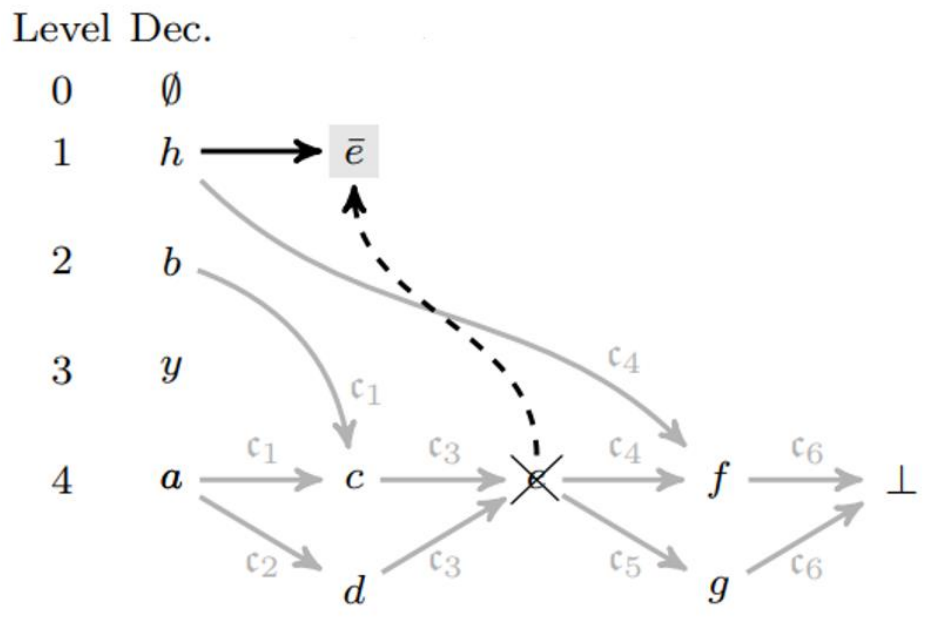


$$\begin{tikzcd}
(\bar{f} \vee \bar{g}) \arrow[dr, dash] & (\bar{e} \vee g) \arrow[d, dash] & (\bar{h} \vee \bar{e} \vee f) \arrow[dd, dash] \\
 & (\bar{e} \vee \bar{f}) \arrow[dr, dash] & \\
 & & (\bar{h} \vee \bar{e})
\end{tikzcd}$$

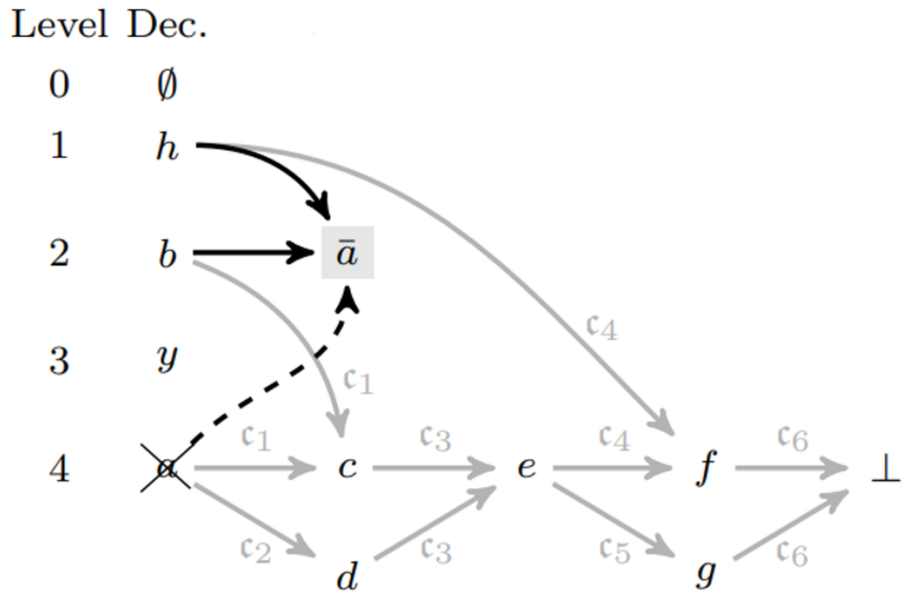
Resolution steps with first UIP learning

CDCL – Conflict Analysis

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



Backtrack with first UIP learnt clause $\bar{h} \vee \bar{e}$

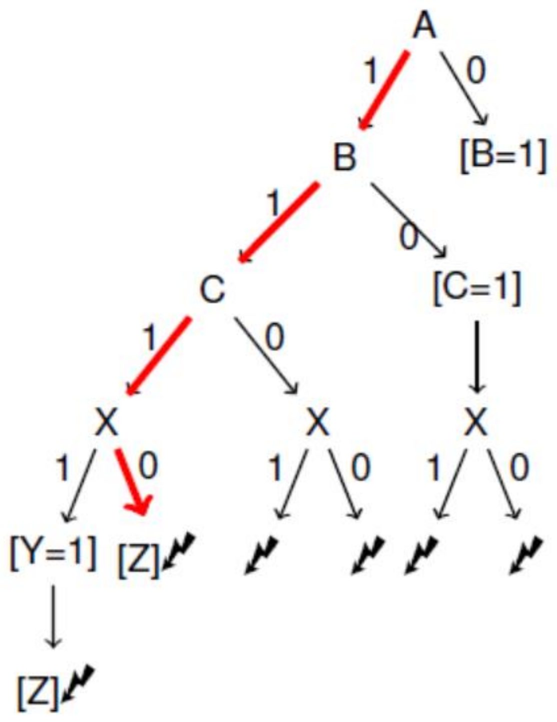


Backtrack with learnt clause $\bar{h} \vee \bar{b} \vee \bar{a}$

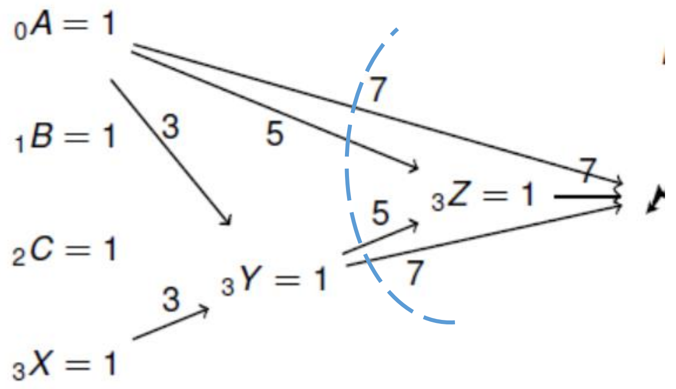
回退到学习子句的第二深（就是除了当前层之外，最深的一层）

Backtracking with Clause Learning

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



Chronological Backtracking



Conflict Analysis

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

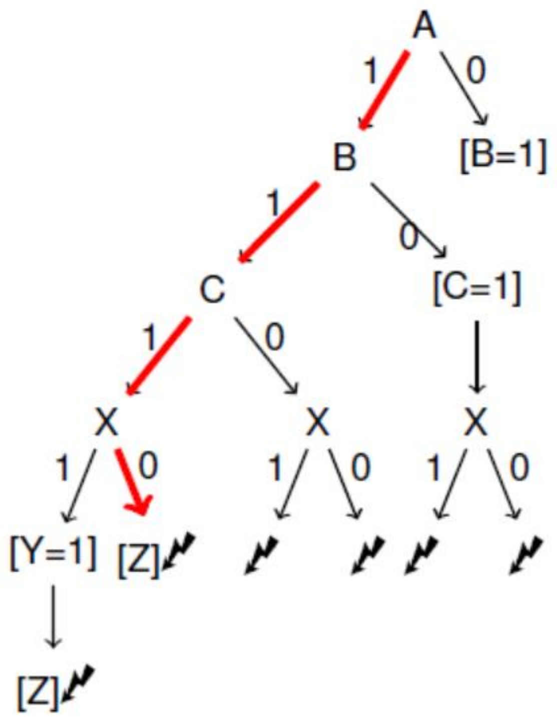
Conflict Clause(1UIP): $\{\neg A, \neg Y\}$

Clause Learning

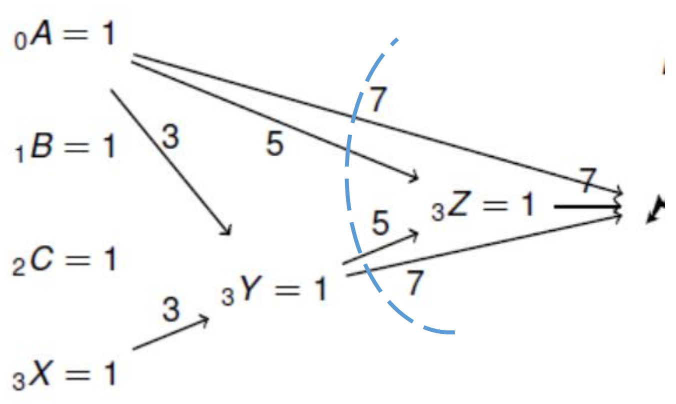
Backtracking with Clause Learning

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

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



Chronological Backtracking

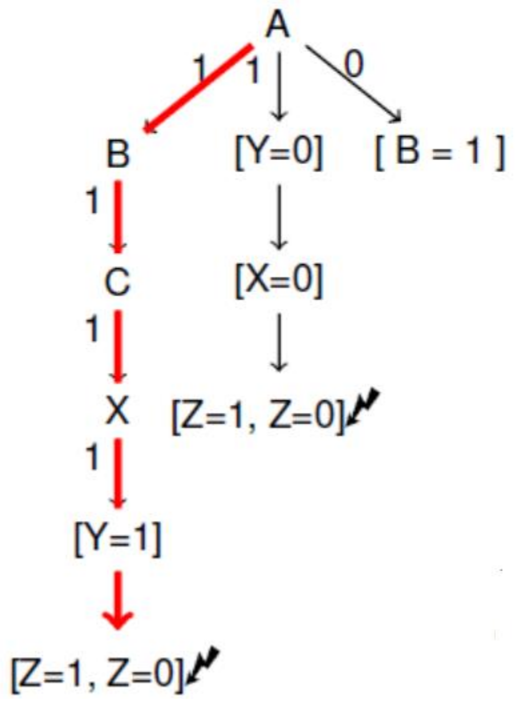


Conflict Analysis

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

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

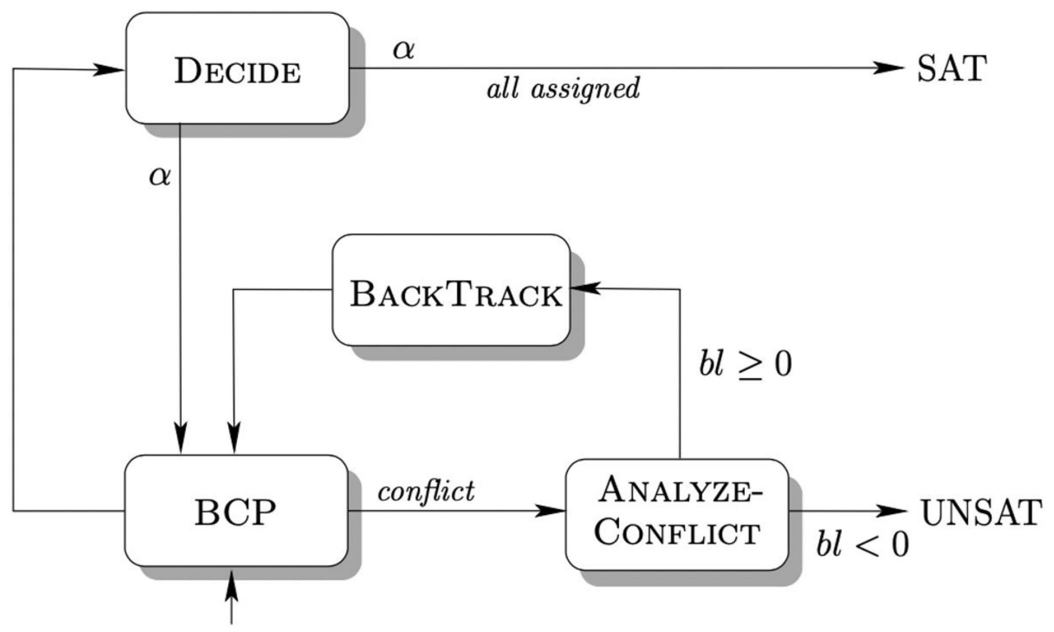
Clause Learning



Non-Chronological Backtracking

CDCL Framework

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

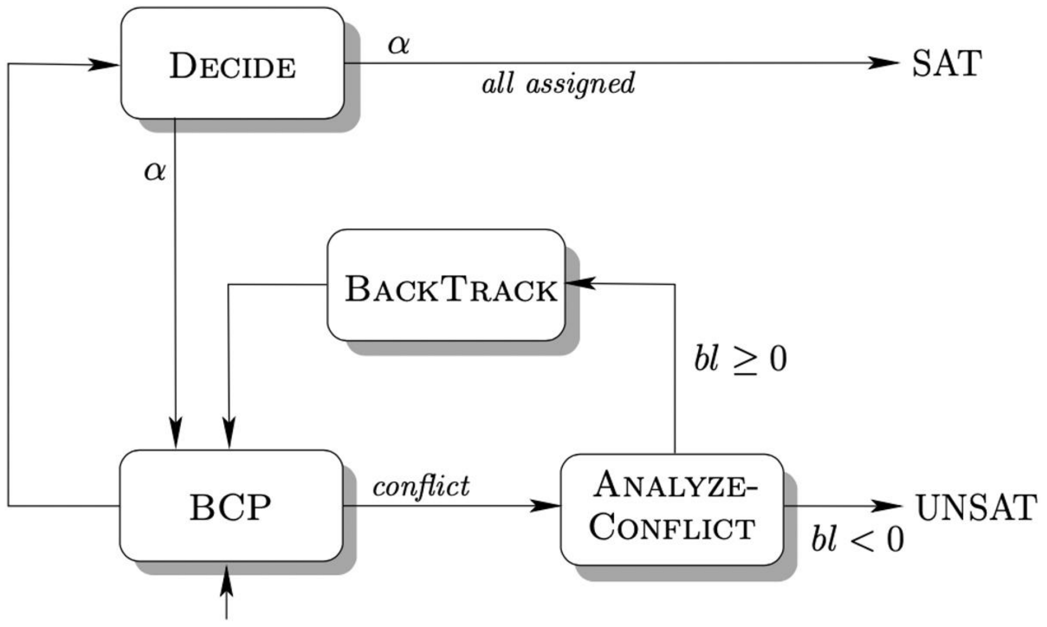


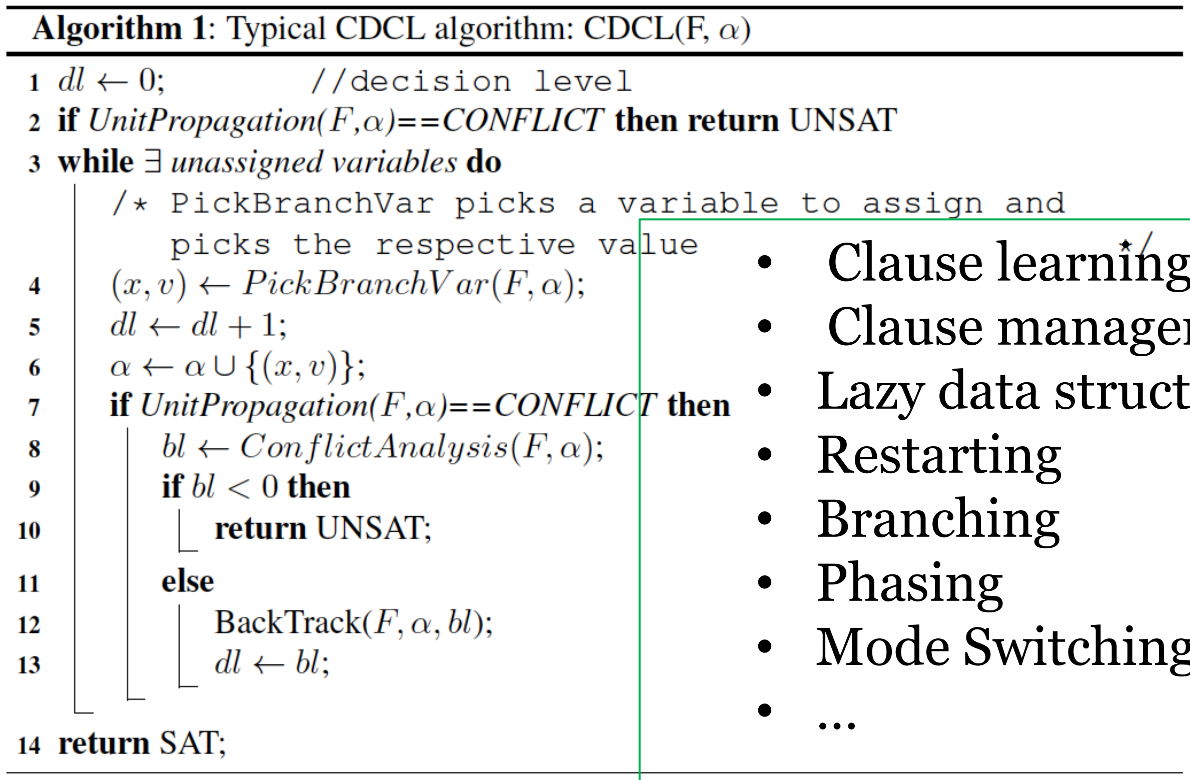
Algorithm 1: Typical CDCL algorithm: CDCL(F, α)

```
dl ← 0;            //decision level
if UnitPropagation(F,α)==CONFLICT then return UNSAT
while ∃ unassigned variables do
    /* PickBranchVar picks a variable to assign and
       picks the respective value                        */
    (x, v) ← PickBranchVar(F, α);
    dl ← dl + 1;
    α ← α ∪ {(x, v)};
    if UnitPropagation(F,α)==CONFLICT then
        bl ← ConflictAnalysis(F, α);
        if bl < 0 then
            return UNSAT;
        else
            BackTrack(F, α, bl);
            dl ← bl;
return SAT;
```

CDCL Framework

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




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

```
dl ← 0;            //decision level
if UnitPropagation(F,α)==CONFLICT then return UNSAT
while ∃ unassigned variables do
    /* PickBranchVar picks a variable to assign and
       picks the respective value */
    (x, v) ← PickBranchVar(F, α);
    dl ← dl + 1;
    α ← α ∪ {(x, v)};
    if UnitPropagation(F,α)==CONFLICT then
        bl ← ConflictAnalysis(F, α);
        if bl < 0 then
            return UNSAT;
        else
            BackTrack(F, α, bl);
            dl ← bl;
return SAT;
```

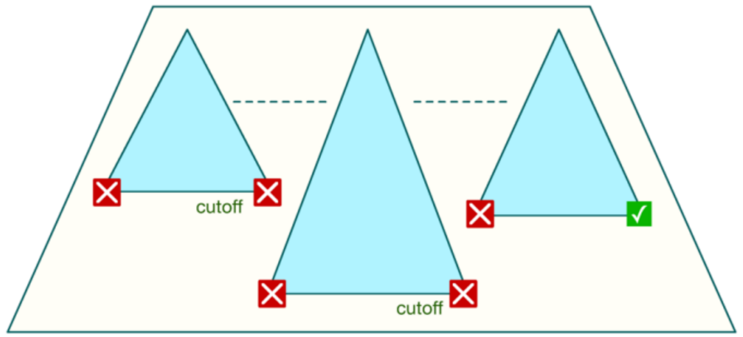
- Clause learning
- Clause management
- Lazy data structures
- Restarting
- Branching
- Phasing
- Mode Switching
- …

CDCL Heuristics – 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 Heuristics – 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.



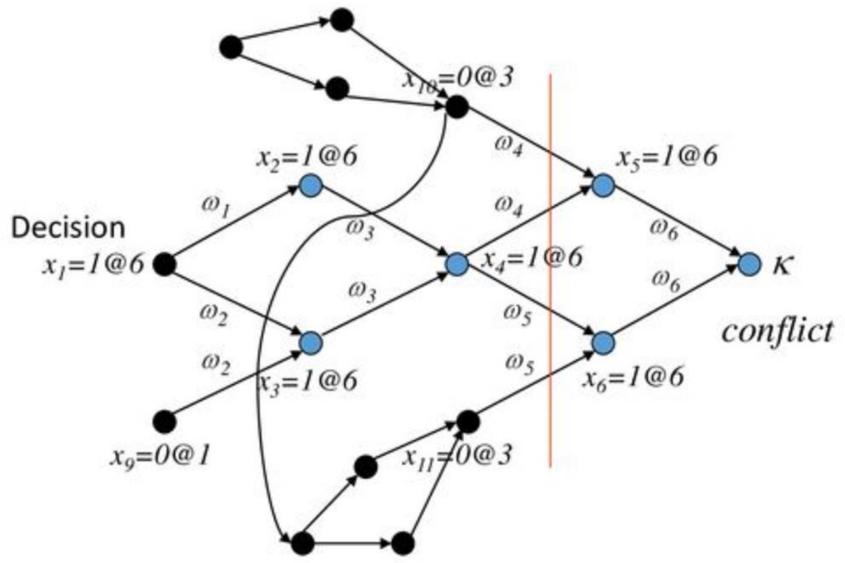
- Restart policies:
  - Luby series: 1 1 2 1 1 2 4 1 1 2 1 1 2 4 8 …
  - Glucose restart (rapid) : When average LBD of some current learnt clauses is great than the average LBD of all learnt clauses. [AudemardSimon,12CP]

- The completeness issue caused by restart
  - To keep completeness of SAT solvers, either restarts must be delayed more and more,
  - or alternatively the number of learnt clauses kept during database reduction needs to be increased

CDCL Heuristics – 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}

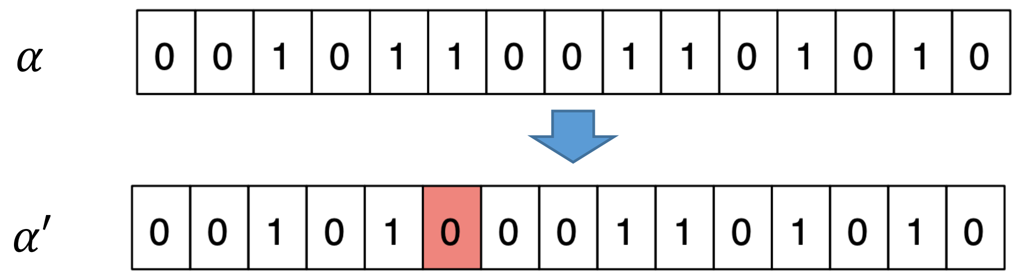
Outline

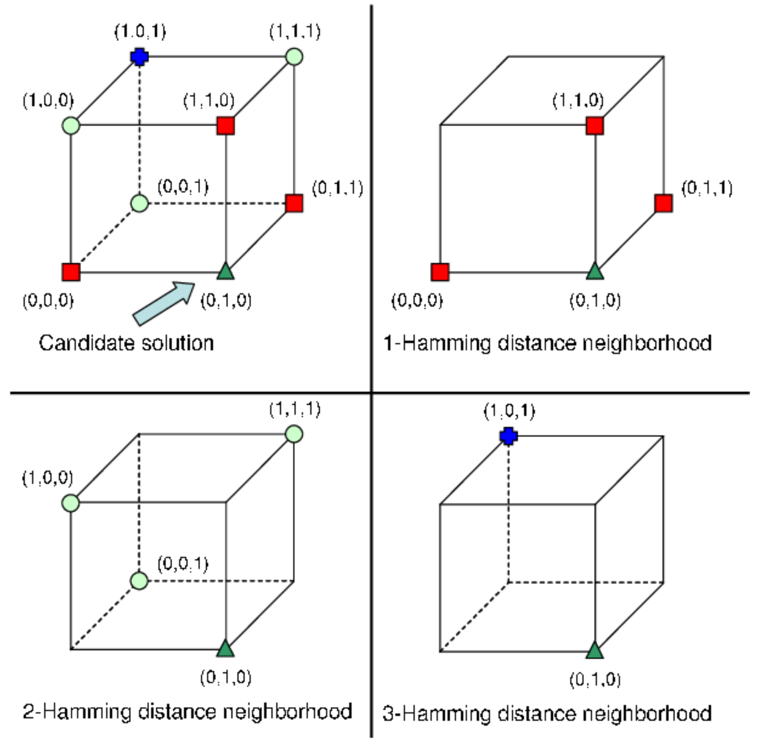
- SAT Basis
- CDCL
- Local Search
- Hybrid SAT Solving

Stochastic 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.





(1,0,1)
(1,1,1)
(1,0,0)
(1,1,0)
(0,0,1)
(0,1,1)
(0,0,0)
(0,1,0)
Candidate solution
1-Hamming distance neighborhood
2-Hamming distance neighborhood
3-Hamming distance neighborhood

Search space of a SAT instance with 3 variables

Local Search (LS) Algorithms

search space S

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

solution set S′ ⊆ S

(SAT: models of given formula)

neighbourhood relation N ⊆ S × S

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

objective function f : S → R+

(SAT: number of clauses unsatisfied under given assignment)

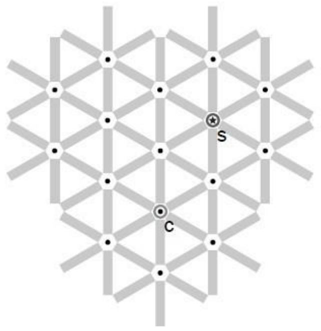
evaluation function g : S → R^+

(g1=number of clauses unsatisfied under given assignment;

g2=total weight of clauses unsatisfied under given assignment, using clause weighting techniques.)

A geometrical viewpoint of Local Search

- the whole search space forms a network
- a local search algorithm starts from an initial position
- iteratively moves from current position to neighbouring position
- use evaluation function for guidance



- important components in local search design
  - Neighbourhood relation (or, move/operator) --- usually just flip a variable in SAT
  - Evaluation function
  - Search strategy

Run a small example

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

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

- S=<000>, N(S)={S1,S2,S3}={<100>,<010>,<001>}
- Evaluation: the number of unsatisfied clauses
- 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 in the value of x.

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

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

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 in the value of x.

$F=\{x_1 \vee \sim x_2,\ x_1 \vee x_2,\ x_2,\ \sim x_1 \vee x_2 \vee \sim x_3\}$		
	assignment	unsatisfied clauses
S	000	$x_1 \vee x_2,\ x_2$
S	100	x_2

- $score(x_1)=g(100) - g(000)=1-2=-1$
- $score(x_2)=g(100) - g(110) =1-0=1$
- $score(x_3)=g(100) - g(101) =1-2=-1$

Score of Variables

$F=\{x_1 \lor \sim x_2,\ \ x_1 \lor x_2,\ \ x_2,\ \ \sim x_1 \lor x_2 \lor \sim x_3\}$		
	assignment	unsatisfied clauses
S	000	$x_1 \lor x_2,\ \ x_2$
S	100	x_2
S	110	None

GSAT: a simple LS based on score

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;
```

Caching based calculation of scores

- The key to implementing many local search SAT algorithms efficiently lies in caching and updating the variable scores.
  - compute all scores when the search is initialized
  - subsequently only update the scores affected by a variable that has been flipped
- After flipping a variable x, only clauses dependent on x can change their satisfaction status.
- Traverse all clauses for which x appears, and update the scores of variables in the clause.

Caching based calculation of scores

For each clause C, denote the number of true literals as true_lit_num;

- Case 1: flipping x makes the literal of x (could be x or $\neg x$) in C become true.
  - 1.1 true_lit_num 0→1.

    E.g. : C: $x \vee y \vee z$, assignment 0 0 0 → 1 0 0 after flipping x
    - before flipping x, clause C contributes 1 to score(y) and score(z);
    - after flipping x, clause C contributes 0 to score(y) and score(z).
    - → thus, $\Delta_C score(y) = -1, \Delta_C score(z) = -1$
  - 1.2 true_lit_num 1→2
    - E.g. C: $x \vee y \vee z$, assignment 0 1 0 → 1 1 0 after flipping x
    - Before flipping, C contributes -1 to score(y) and 0 to score(z);
    - After flipping, C contributes 0 to all variables in C.
    - →thus, only update score(y), $\Delta_C score(y) = +1$.

Critical true literal: y

- What about the change on score(x)?
  - Simply, score(x) turns to -score(x)

Scoring Functions

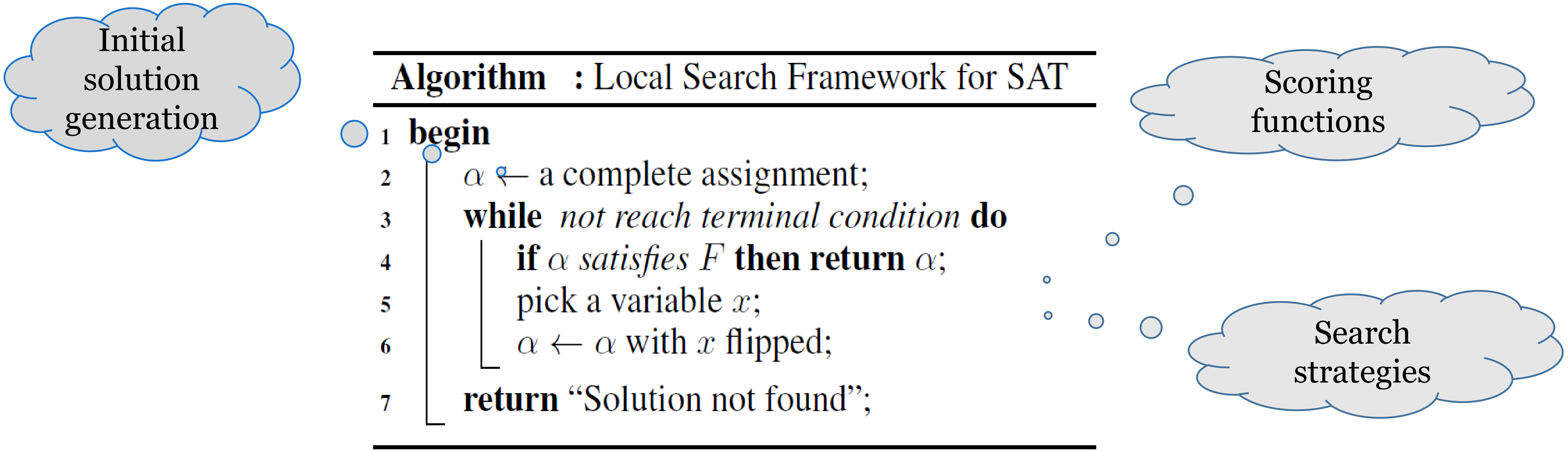
A basic Scoring Function

- 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)

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
- …

Stochastic Local Search for SAT



Algorithm : Local Search Framework for SAT

```
begin
    $\alpha \leftarrow$ a complete assignment;
    while not reach terminal condition do
        if $\alpha$ satisfies $F$ then return $\alpha$;
        pick a variable $x$;
        $\alpha \leftarrow \alpha$ with $x$ flipped;
    return "Solution not found";
```

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.

More Scoring functions

- $A^{\mathrm{score}(x)}$
- $\mathrm{score}(x)^{B}$
- …

Dynamic Scoring functions

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

Satisfaction degree

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

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

Second Level Scoring Functions

- Second Level Scoring Functions
  - $make_2$(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

Clause weighting

- Evaluation function： $g(F,\alpha) \coloneqq \Sigma_{c \in UC(F,\alpha)} w(C)$
  - Date back to the Breakout method (1993): increases the weight of each unsatisfied clause by one when reaching local optima.
  - The Breakout method allows unrestricted weight growth during.
- Modern clause weighting usually have a “smoothing” mechanism to decrease clause weights.

Strategies to deal with cycling

- 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.

Strategies 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 [1989].

- Configuration checking
  - Forbid a variable to be flipped if its configuration has not changed after its last flip [2011].

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

- Check whether a variable is tabu, by using time stamp, so that we do not need a tabu list.

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

The tt parameter is instance dependent, and needs to be fine tuned.

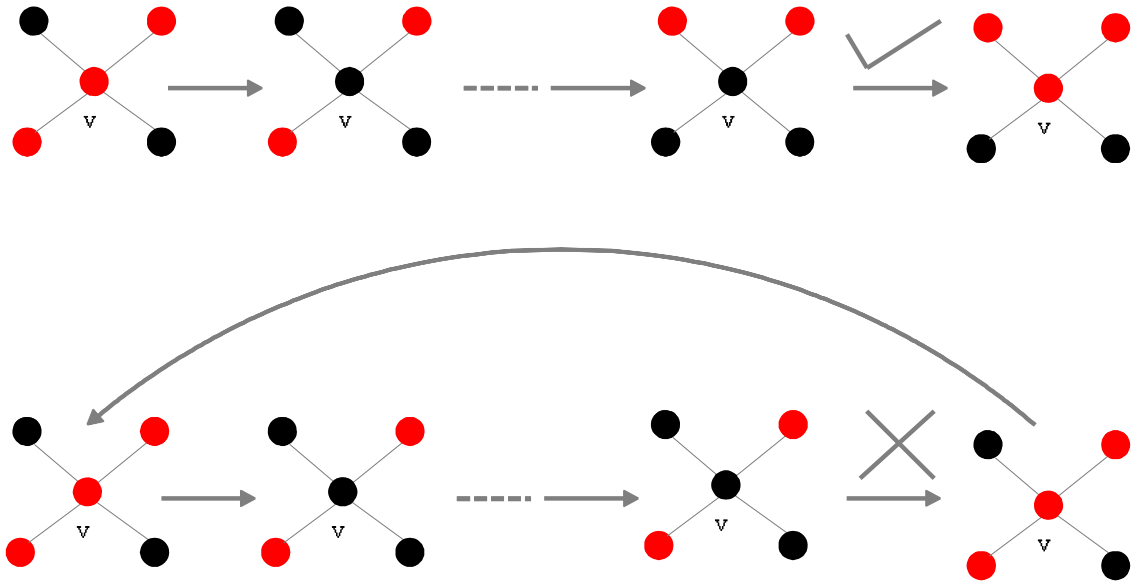
Configuration Checking (CC)

- 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).

Configuration Checking for SAT

- $\boldsymbol{N(x)} = \{y|y \textit{ and } x \textit{ occur in at least one clause}\}$
- **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 $y \in \mathrm{N}(x)$, 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 $O(n) + O(\Delta(F))$

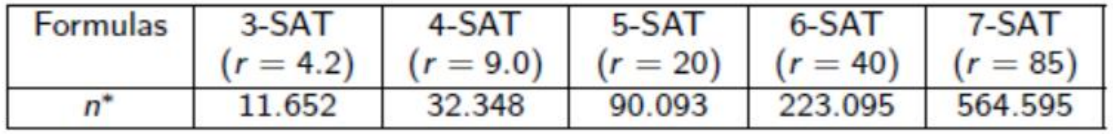
- Indeed, the number of candidate variables for flipping is much smaller than n.

When does CC (not) work?

The inequality of effectiveness

Theorem: For random k-SAT model $F_k(n, r)$, the neighborhood based CC becomes trivial that forbids only one variable when $ln(n-1) < \frac{k(k-1)r}{n-1}$ (r is the clause-variable ratio).

Let $\boldsymbol{f(n) = ln(n-1) - \frac{k(k-1)r}{n-1}}$, $\boldsymbol{f(n)}$ monotonically increase with n (n>1).
Thus, f(n)<0 iff $\boldsymbol{n \leq \lfloor n^* \rfloor}$, where $\boldsymbol{f(n^*) = 0}$



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

When $\boldsymbol{n \leq \lfloor n^* \rfloor}$, the CC becomes trivial and ineffective.
The inequality rarely happens in real world instances.

Types of local search SAT solvers

Local search for SAT mainly fall into two types:

- focused local search: always picks the flip variable from an unsatisfied clause.
- two-mode local search: switches between intensification mode(usually using global search) and diversification mode (usually using focused local search).
- There is also a significant line of research concerns about weighting techniques.

Focused Random Walk (FRW)

- In this type of random walk, 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).*

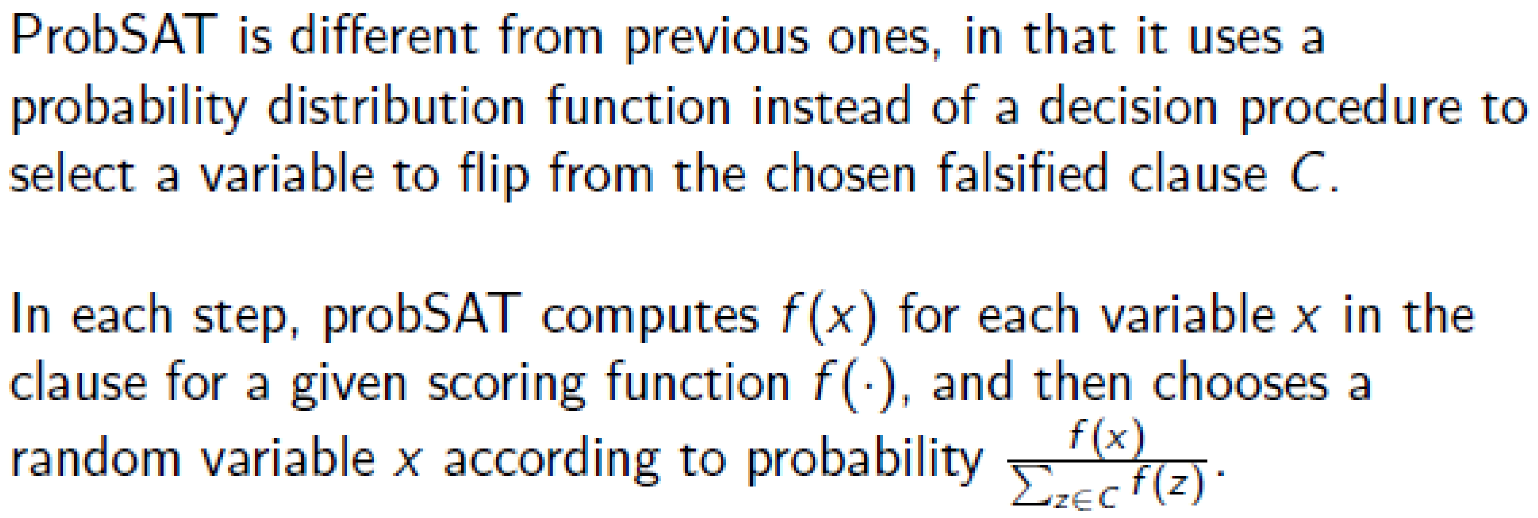
Algorithm 3: WalkSAT

```
$s \leftarrow$ a randomly generated truth assignment;
while not reach terminal condition do
    if $s$ satisfies $F$ then return $s$;
    $c :=$ an unsatisfied clause of $F$ chosen at random;
    if $\exists$ variable $x \in c$ with $break(x) = 0$ then
        $v := x$, ties breaking randomly;
    else
        With probability p, $v :=$ a variable in $c$ chosen at random;
        With probability 1-p, $v :=$ a variable in $c$ with the smallest break, ties
        breaking randomly;
    $s := s$ with $x$ flipped;
return "Solution not found";
```

Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In Proceeding of AAAI 1994.

Focused Random Walk (FRW)

- ProbSAT



ProbSAT is different from previous ones, in that it uses a probability distribution function instead of a decision procedure to select a variable to flip from the chosen falsified clause C.

In each step, probSAT computes $f(x)$ for each variable x in the clause for a given scoring function $f(\cdot)$, and then chooses a random variable x according to probability $\frac{f(x)}{\sum_{z \in C} f(z)}$.

Two-mode local search

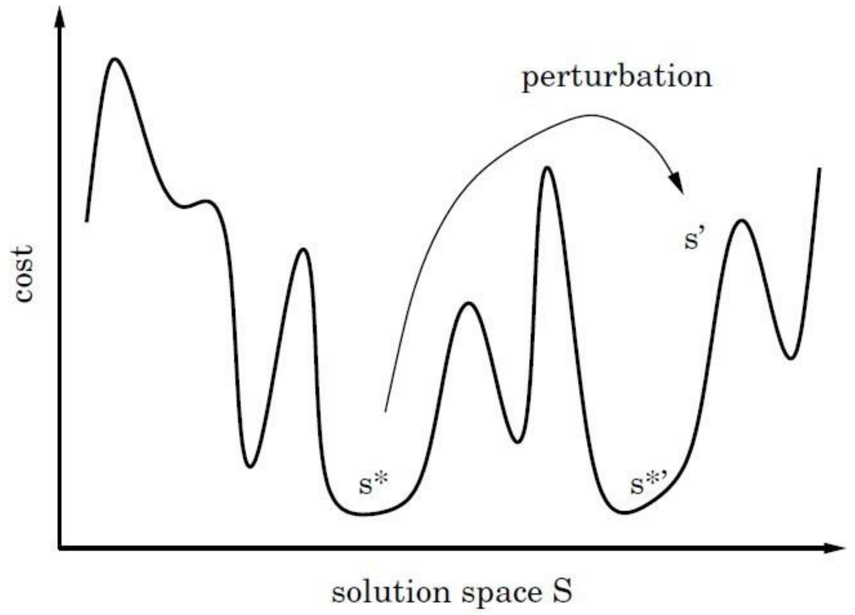
Any two-mode local search solver can be characterized by specifying the following three components.

- intensification mode;
- Diversification (random walk) mode;
- switching rule.

Two-mode local search

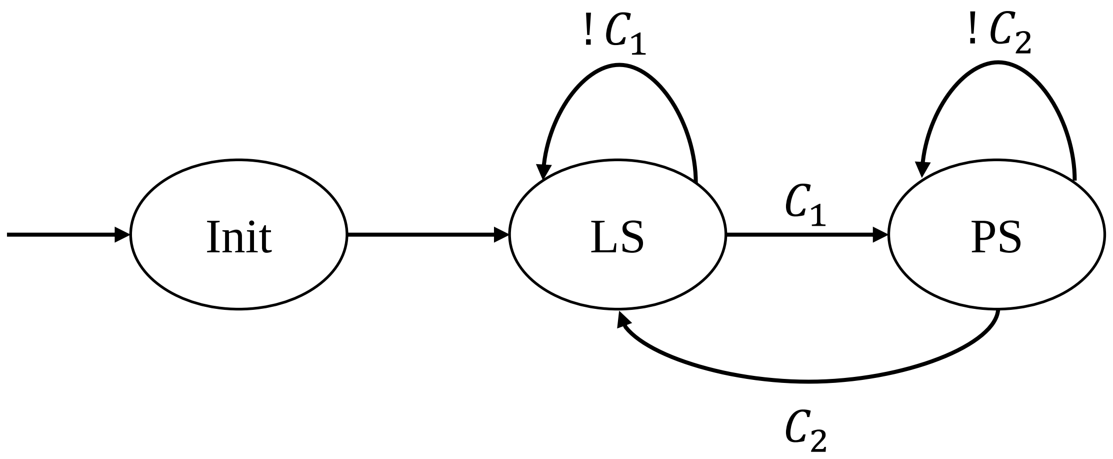
Iterated Local Search (ILS)

- Key Idea: Use two types of search steps:
    - local search steps for reaching local optima as efficiently as possible (intensification, exploitation)
    - perturbation steps for effectively escaping from local optima (diversification, exploration).



ILS trajectories can be seen as walks in the space of local minima of the given evaluation function.

Modelling LS Methods



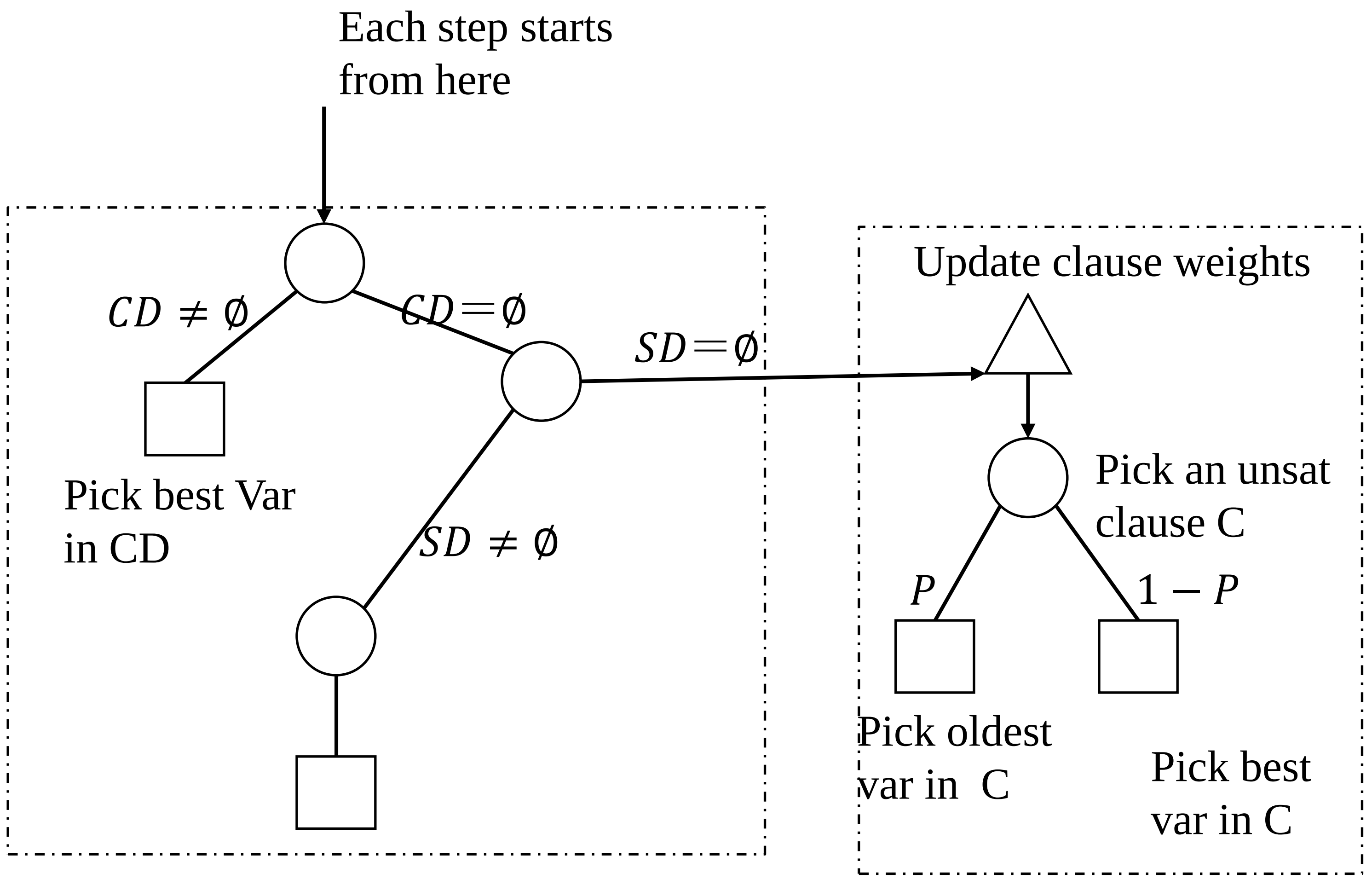
```
• S := a random complete assignment;
• while (!termination condition)
    • if (exsit variables with positive score)
        • x:= a variable with positive score;
        • S := S with x flipped.
    • else
        • make changes on S randomly.
```

Two-mode local search - CCAnr

- Configuration checking (CC)
- Smoothed Weighting (SW)
- Aspiration

CD:={x|score(x)>0 and CC[x]=1}.

SD:={x|score(x) > $\overline{w}$}



Hybrid Solving – CDCL Solving + SLS Sampling

CaiZhang, ***SAT*** 2021.

CDCL focuses on a local space in a certain period

-->Better to integrate reasoning techniques

SLS walks in the whole search space

->Better at sampling

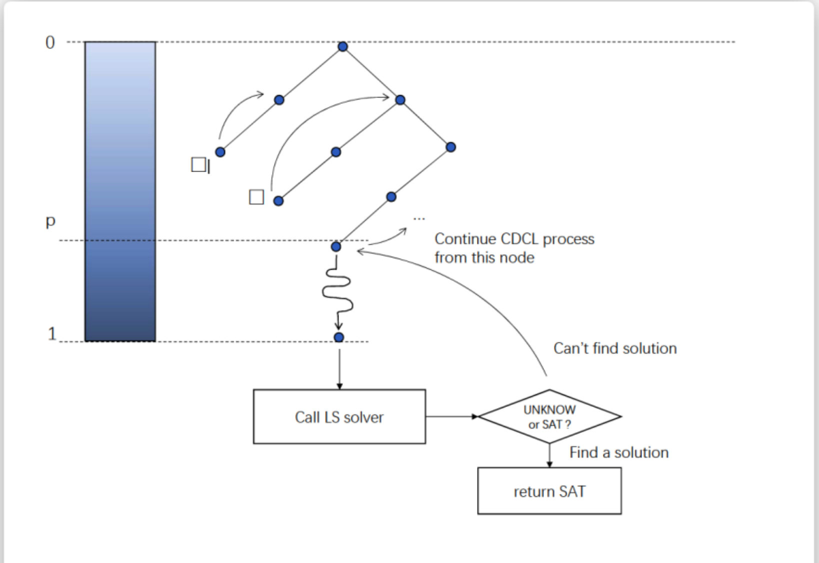


Fig. 1. Overall Procedure of Relaxed CDCL

SLS sampling ←→ CDCL solving
Boosting CDCL with SLS information

Plug SLS into a CDCL solver

- Calling SLS on promising branches
- Filter similar branches

Hybrid Solving – Phase Resetting with SLS

- Phase selection is an important component of a CDCL solver.

- Phase resetting with SLS assignments:
  - resets the saved phases of all variables according to a probability distribution of SLS assignments.
  - Make use of the top promising branch and top SLS results.

Hybrid Solving – Branching with SLS

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

Branching with SLS conflict frequency:

- calculate the conflict frequency in SLS
- Transform conflict frequency into the branching metrics.

Others...

- Preprocessing / Inprocessing (Interleave search and preprocessing)
  - Clause Distribution
  - 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

SAT, a generic methodology...

Thank you!