SAT Solvers

(Meena Mahajan
meena@imsc.res.in)

Partition $\{1, \ldots, 10\} = [10]$ into two sets R & B such that neither set contains a "Pythagorean triple" $a^2 + b^2 = c^2$ (a, b, c)

(3, 4, 5)
(6, 8, 10)
(5, 12, 13)
(9, 12, 15)
(8, 15, 17)
(12, 16, 20)

1, 2, 3, 4, ~~10~~ 10, 11   5, 6, 7, 8, 9

Partition [20]

[100] —— 2^{99} ways

[1000]

[10,000] . . . . .

Does such a partition exist? for all N.

N: Yes ⇝ convert a proof to a partition
No ⇝?

No: Fails at 7,825 ⟶ 2016 [Heule Kullmann Marek]

- Solved using computer
  - Correct? Produced a proof.
    - 200 TeraBytes long skeleton
    - who can check this?
    - Proof was computer-verified
      - Download, reconstruct, verify → 30,000 cPU hours
    - Verifier Correct?
      - ↳ Compact, simple; Formally Verified

(The Science of Brute Force)
↓
2^{999} ways?

How did HKM use computers to solve the Boolean Pythagorean triple problem?

BPT

↳ Encode BPT[N] as an instance of SAT,
& they used a SAT solver.

Variables $x_1, x_2, \ldots, x_N$ ⟶ values True / False
1 / 0
Red / Blue

such that

$\forall$ PT (a,b,c) (for all)

- at least one of x_a, x_b, x_c True
- " " " " " " False

x_a OR x_b OR x_c

$\bar{x}_a$ OR $\bar{x}_b$ OR $\bar{x}_c$

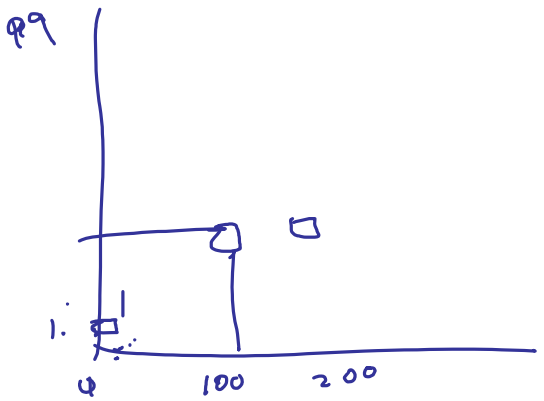
∴ $(x_a \vee x_b \vee x_c)$ ⟶ clause

∴ $(\bar{x}_a \vee \bar{x}_b \vee \bar{x}_c)$ ⟶ clause

Is the collection of clauses

$$\left\{ \begin{array}{l} x_a \vee x_b \vee x_a \\ \bar{x}_a \vee \bar{x}_b \vee \bar{x}_c \end{array} \right\} \quad 1 < a < b < c \leq N$$

simultaneously satisfiable?



For $N \leq 7824$, colorings with R, B, W

SAT solver

i/p F: Formula (collection of clauses)

o/p Yes, $a_1 \ldots a_n \in \{0,1\}^n$

OR

No, ~ proof that F is really unsat

(I) Brute force Algo: Try out all assignments

(II) " " Algo: Backtracking

① $x \vee z$

② $y \vee \bar{z}$

③ $x \vee \bar{y} \vee u$

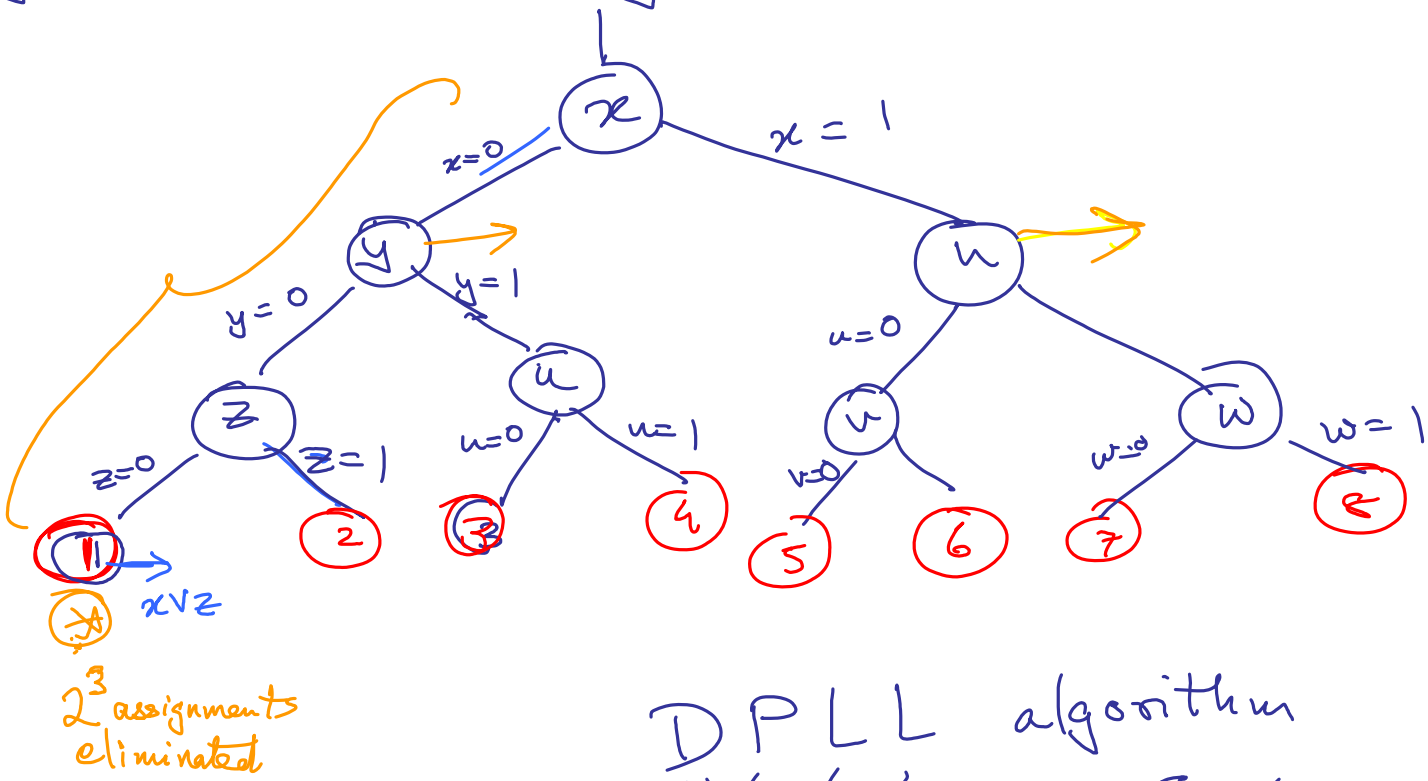
④ $\bar{y} \vee \bar{u}$

⑤ $u \vee v$

⑥ $\bar{x} \vee \bar{v}$

⑦ $\bar{w} \vee w$

⑧ $\bar{x} \vee \bar{w} \vee \bar{w}$



2^3 assignments eliminated

DPLL algorithm

50s, 60s

- SAT Revolution (90s)

  CDCL algorithms.

  (1) - Boolean Constraint Propagation (Unit Propagation)

  ~ 99% of the "time"/"assignments" in modern solvers are BCP

  Variable Decision / Value Heuristic

  VSIDS [Variable State Independent Decaying Sum]

  - reward variables that have participated more recently in conflicts
  - "damping"

CDCL

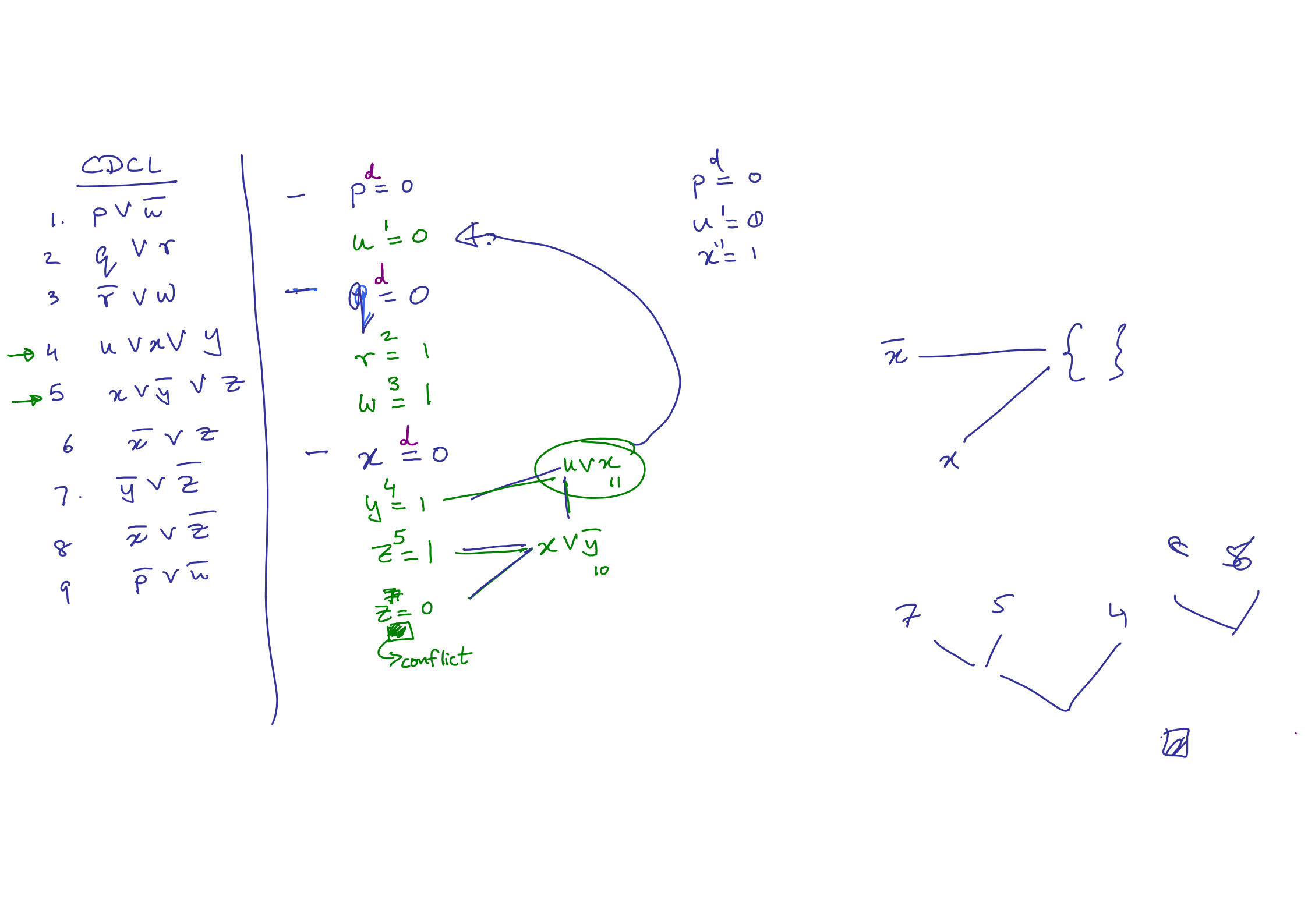
(2) Conflict Analysis (prunes larger space)

Conflict-Driven Clause Learning

Backtrack Jump

Learning Scheme → Heuristic

(3) Reset Heuristic


CDCL
1. p ∨ w̄
2 q ∨ r
3 r̄ ∨ w
4 u ∨ x ∨ y
5 x ∨ ȳ ∨ z
6 x̄ ∨ z
7. ȳ ∨ z̄
8 x̄ ∨ z̄
9 p̄ ∨ w̄
p d = 0
u 1 = 0
q d = 0
r 2 = 1
w 3 = 1
x d = 0
y 4 = 1
z 5 = 1
z = 0
conflict
u ∨ x
11
x ∨ ȳ
10
p d = 0
u 1 = 0
x 1 = 1
x̄
{ }
x
7
5
4

Today, CDCL-based solvers can handle millions of variables

SAT contests

Heuristics, Tricks —— Decision, Learning, Restart

- Stack / Queue
- Speed up BCP : (2) Watched literals
- Forgetting some learnt clauses