SAT-Solving

- Though Boolean (propositional/sentential) logic is much less expressive than FOL, many AI problems involve constraint-solving, where the constraints are easily expressed as boolean formulas.

  e.g., class scheduling (room/instructor/student availab'y)
  conference paper refereeing
  hardware/software verification
  propositional planning

- We can even handle quantified knowledge, if it is function-free (after Skolemization), by "grounding" quantified formulas (e.g. planning)

- Millions of variables can currently be handled in industrial-scale benchmark problems, & speeds have more or less been doubling every year.

phase transition: $c = 4.256v + 58.26v^{-2/3}$ (Crawford & Anton '96)

Refs: p77-83 Brachman & Levesque; Section 7.6 Russell & Norvig; Malik & Zhang, "Boolean Satisfiability:...", CACM 52(8), Aug. 2009, 76-82.

Also J. Fichte, CACM June 2023

Most common algorithm: DPLL (Davis, Putnam, Logemann, Loveland)

Idea, for a set of clauses (CNF):

Repeat

Choose a variable & set it to 1 or 0;
(Prefer variables of unit clauses & ones that appear with only one sign; the choice of 1 or 0 should make at least 1 clause true)
Simplify the clauses accordingly, returning "SATISFIABLE" if no clauses remain;
Backtrack to an earlier choice if an inconsistency appears, or return "UNSATISFIABLE" if no choices remain

Until all clauses are satisfied

e.g., $(\neg x_1 \vee \neg x_2), (x_1 \vee x_2 \vee \neg x_3), (\neg x_1 \vee x_3 \vee \neg x_4), (x_1 \vee x_4), (x_1 \vee x_5)$

Choose $x_5 = 1$ (appears positively only, in last clause)

So: $(\neg x_1 \vee \neg x_2), (x_1 \vee x_2 \vee \neg x_3), (\neg x_1 \vee x_3 \vee \neg x_4), (x_1 \vee x_4)$

Choose $x_2 = 1$ (not the best choice but OK)

So: $\neg x_1, (\neg x_1 \vee x_3 \vee \neg x_4), (x_1 \vee x_4)$

Choose $x_1 = 0$ (necessarily)

So: x_4

Choose $x_4 = 1$ (necessarily), $x_3 = 1$

So: SATISFIABLE (no clauses left)

So: SATISFIABLE (no clauses left)

Note: A satisfying assignment is easily extracted from a satisfiable clause set. Use arbitrary values for variables that didn't receive a value.

A major improvement from the mid-90's:

conflict-driven learning (of new clauses that must be true if there is a satisfying assignment)

with

non-chronological backtracking (jumping backward over more recent choice points, to a point where a decision relevant to a detected conflict was made.)

Example from Malik & Zhang (with 3 clauses added):

Clauses

$\neg x_1 \vee x_4$
$x_1 \vee x_4$
$x_1 \vee \neg x_3 \vee \neg x_8$
$x_1 \vee x_8 \vee x_{12}$
$\neg x_2 \vee x_{10}$
$x_2 \vee x_{11}$
$x_3 \vee x_{10}$
$\neg x_7 \vee \neg x_3 \vee x_9$
$\neg x_7 \vee x_8 \vee \neg x_9$
$x_7 \vee x_8 \vee \neg x_{10}$
$x_7 \vee x_{10} \vee \neg x_{12}$

Choices

$x_1 = 0$, so $x_4 = 1$ ← DPLL would choose $x_4 = 1$ right away (or $x_4 = 0$), but...
$x_3 = 1$, so $x_8 = 0$, $x_{12} = 1$
$[x_7 = 0]$
$x_2 = 0$, so $x_{11} = 1$
$x_7 = 1$, so $x_9 = 1, 0$

caused by choices $x_3 = 1$, $x_7 = 1$, $x_8 = 0$

so assert $\neg x_3 \vee \neg x_7 \vee x_8$

"learned" clause

back to earliest point at which one of the var's of the learned clause was determined

After the first 2 steps, since $x_3 = 1$ and $x_8 = 0$, we must have $x_7 = 0$, so we jump back to step 2
(step 3 is irrelevant to the conflict)

Other improvements to DPLL:

- "two-literal watching" (efficient unit clause usage)
- local search (in the "vicinity" of the current assignment - see below (LS))

Other algorithms in the text: tableau (TAB), LS and GSAT.

backtrack control is handled by recursion

TAB: Similar to DPLL, but we focus on some remaining clause, & choose one of its literals (not yet tried) to be true (=1) (backtracking to another of its literals, not yet chosen, if the rest of the procedure fails); eliminate clauses thereby made true, and recurse till all clauses are true (eliminated) or no choices remain

LS: Searches within a growing (upper bound $n/2$) Hamming distance of 2 or more starting assignments } See B&L p 80-81

GSAT: Starting with arbitrary assigment, keep flipping the truth value of a variable that will increase the number of true clauses as much as possible. Use a limit on the number of flips, & if necessary start with a new random assignment.

Doesn't guarantee finding a satisfying assignment if one exists, & doesn't prove unsatisfiability.