SAT-Solving

- Though Boolean (propositional) 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 available), 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.

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

---

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

Idea, for a set of clauses (CIFL):

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., \((x_1, \neg x_2, x_3), (x_4, x_2, \neg x_3, x_4), (x_1, x_3), (x_1, x_4)\)
Choose \(x_2 = 1\) (appears positively only, in last clause)
So: \((x_1, \neg x_2, x_3), (x_4, x_2, \neg x_3, x_4), (x_1, x_4)\)
Choose \(x_1 = 1\) (not the best choice but OK)
So: \((\neg x_1, \neg x_2, x_3), (x_4, x_4)\)
Choose \(x_4 = 0\) (necessarily)
So: \(x_4\)
Choose \(x_3 = 1\) (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               | Choices         | DPLL would choose x_0 = 1 right away, (or x_0 ≠ 1) but...
|-----------------------|-----------------|--------------------------------------------------------
| ¬x_1 ∨ x_4            | x_1 = 0, so x_4 = 1 | [x_7 = 0] in red                                        |
| x_1 ∨ x_4             | x_2 = 1, so x_5 = 0, x_4 = 1 | caused by choices x_3 = 1, x_7 = 1, x_8 = 0 so assert ¬x_3 ∨ ¬x_7 ∨ x_8 
| x_1 ∨ x_2 ∨ x_3 ∨ x_8 | x_2 = 0, so x_11 = 1 | back to earliest point at which one of the users of the learned clause was determined |
| x_1 ∨ x_8 ∨ x_12      | x_7 = 1, so x_9 = 1, 0 |
| x_2 ∨ x_11            |                           |
| x_5 ∨ x_10            |                           |
| x_6 ∨ x_7 ∨ x_3 ∨ x_4 |                           |
| ¬x_7 ∨ x_8 ∨ x_9      |                           |
| x_7 ∨ x_10 ∨ x_12     |                           |

After the first 2 steps, since x_2 = 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.

**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 Hamming distance of 2 or more starting assignments.

**GSAT:** Starting with arbitrary assignment, 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.