# Discussion Problems: "SAT Solvers"
# Declarative Methods (JHU 600.432/632)
# Prof. Jason Eisner

1. [14 points] Recall that the zChaff SAT solver is basically an implementation of the DPLL algorithm, but with some modifications for efficiency. Suppose it discovers during unit propagation that $X$ must be true. What bookkeeping is associated with setting $X = 1$ by unit propagation? *(circle <u>one</u> choice from each group)*

   (a) The decision level (INCREASES, DECREASES, STAYS THE SAME).

   (b) If $X = 0$ already, then the solver (BACKTRACKS, RETURNS "UNSAT," DOESN'T NOTICE YET, THIS CASE CANNOT ARISE).

   (c) The solver visits (ALL, SOME, NONE) of the clauses that contain $X$.

   (d) The solver visits (ALL, SOME, NONE) of the clauses that contain $\neg X$.

   (e) The number of clauses visited is (SOMETIMES MORE, ALWAYS THE SAME, SOMETIMES LESS) than if $X = 1$ had been set by an explicit assignment.

   (f) When the solver visits a clause of length $k \geq 2$, it checks the current values of at least $(0, 1, 2, k-1, k)$ and at most $(0, 1, 2, k-1, k)$ other literals mentioned in the clause.

2. [8 points] You are using an ordinary (Davis-Logemann-Loveland) backtracking CNF-SAT solver. Show how the following collection of clauses changes when you assign $D = 0$ and perform unit propagation.

   $A \vee B \vee C$
   $A \vee \neg B \vee \neg C$
   $B \vee D$
   $C \vee \neg D$

3. [4 points] The previous problem told you to try $D = 0$ as the initial assignment. What initial assignment might be tried instead by the Jeroslow-Wang "most satisfying" heuristic? (Specify both the variable and the value. As usual, feel free to explain your answer.)

4. [5 points] DPLL is usually implemented (e.g., in zChaff) by using a stack of assignments. For example, when you make a decision to assign $C = 0$, this goes on the stack. This lets you later backtrack, undo $C = 0$, and try the alternative $C = 1$ instead. On the class slides, $C = 0$ was shown as a yellow stack entry ("first assignment") that was replaced by a red stack entry for $C = 1$ ("second assignment").

   But $C = 1$ is the last value we are trying for $C$, so if we backtrack to that red stack entry, we'll just backtrack right on past it. So perhaps we don't need to have it on the stack at all.

   Thus, here is a suggested optimization that is intended to eliminate "tail recursion": don't bother to push red entries on the stack. When we backtrack to $C = 0$ and try the new assignment $C = 1$, we should pop the yellow $C = 0$ entry from the stack, and change our assignment from $C = 0$ to $C = 1$, but don't bother to push a new red $C = 1$ entry. This saves us the work of pushing $C = 1$ now and also the work of popping it later.

   Would this optimization work, or is there a bug? Explain.

5. (a) [3 points] You have a SAT problem expressed as a conjunction of constraints (also known as clauses). What must each constraint look like if your problem is in Conjunctive Normal Form (CNF)?

   (b) [4 points] Suppose you have a constraint $X \leftrightarrow Y$, where $X$ and $Y$ are variables. Convert this constraint to CNF. Make your CNF version as small as possible. (Shorter answers will get more credit.)

   (c) Intuitively, if a larger SAT problem includes the constraint $X \leftrightarrow Y$, or your equivalent CNF version from problem 5b, then assigning a value to $X$ should allow you to immediately assign a value to $Y$ (or vice-versa). And most solvers will do roughly that.

      i. [2 points] In systematic CNF-SAT solvers (like zChaff), what step makes $Y = 0$ get assigned quickly once $X = 0$ is assigned?

      ii. [3 points] If a systematic CNF-SAT solver were forced to skip that step, what variable ordering heuristic would still make $Y = 0$ get assigned quickly once $X = 0$ is assigned? (There may be more than one answer.)

iii. [3 points] Actually, the previous question seems odd. Variable ordering is about choosing $Y$, not the particular assignment $Y = 0$. Nonetheless, a good variable ordering heuristic (without any value ordering heuristic) *will* indeed ensure that $Y = 0$ gets assigned quickly once $X = 0$ is. Explain why.

iv. [3 points] Will GSAT assign $Y = 0$ quickly once $X = 0$ is assigned? Justify your answer.

6. You wish to apply one step of variable elimination to eliminate $A$ from this CNF-SAT formula, $\Phi$:

$$(A \vee B) \wedge (A \vee C) \wedge (\neg A \vee D) \wedge (X \vee Y)$$

(a) [3 points] In the formula $\Phi$ above, circle the constraints that you must join.

(b) [4 points] Write the result of this step of variable elimination as a new CNF-SAT formula, $\Psi$.

(c) [4 points] If you find a satisfying assignment for the new formula $\Psi$, how would you then solve your original problem $\Phi$?

(d) [2 points] In what sense does a variable elimination step make your problem smaller?

(e) [2 points] In what sense might it make your problem bigger?

7. A stochastic SAT solver just keeps flipping variables. Its speed depends crucially on how fast we can pick the next variable to flip.

The WalkSAT algorithm works on CNF formulas. Recall that to implement WalkSAT, we (a) randomly choose a currently unsatisfied clause $C$. Then, (b) for each literal $L$ in $C$, we must compute a "break score" $b_L$—the number of clauses in which $L$ is the only true literal. If we flipped $L$, we would fix the clause $C$ (and perhaps others), but we would break $b_L$ clauses.

Note that (a) only cares about unsatisfied clauses (0 true literals), whereas (b) only cares about unit clauses (1 true literal). Therefore, we can play some tricks to avoid visiting clauses that are *far from being unit* (several true literals)—just as zChaff does when detecting unit or unsatisfied clauses.

Specifically, as in zChaff, we can say that the first two literals in each clause are "watched." We'll swap literals within a clause as needed to ensure that the first two literals are true if possible.

For each literal $L$ that is currently false, we can maintain 2 vectors (along with their lengths):

- A vector $\mathcal{U}_L$ of all unsatisfied clauses containing $L$.
- A vector $\mathcal{F}_L$ of all unit clauses containing $L$.

And for each literal $L$ that is currently true, we can maintain 2 vectors (along with their lengths):

- A vector $\mathcal{B}_L$ of all unit clauses containing $L$. (These clauses have $L$ as the only true literal, which is watched.)
- A vector $\mathcal{W}_L$ of all non-unit clauses in which $L$ is watched. (These clauses have at least two true literals, of which two are watched, including $L$.)

Furthermore, we maintain

- A vector $\mathcal{U}$ of all unsatisfied clauses. This is the union of the sets $\mathcal{U}_L$.

(*Remark:* An unsatisfied clause of length $k$ appears in $k+1$ different vectors ($\mathcal{U}_L$ and $\mathcal{U}$), and a unit clause appears in $k$ different vectors ($\mathcal{B}_L$ for the true literal and $\mathcal{F}_L$ for the false literals). However, all other clauses only appear in 2 vectors ($W_L$ for their two watched literals), which suggests that they are rarely visited.)

In giving runtimes below, assume that *all* clauses have length $k$, i.e., we have a pure $k$-CNF-SAT problem. Your runtimes should be in terms of $k$ and the vector lengths such as $|\mathcal{U}_L|$.

(a) [3 points] How do we randomly choose an unsatisfied clause $C$? How long does this take?

(b) [3 points] How do we compute the break score $b_L$? How long does this take?

(c) [3 points] How does WalkSAT decide which literal $L$ to flip?

(d) [4 points] If we flip $X$ from 0 to 1, the false literal $X$ becomes true, and the true literal $\neg X$ becomes false. This requires us to make various updates to the vectors and the clauses they point to. The total runtime of these updates turns out to be $O(k \cdot (|\mathcal{U}_X| + |\mathcal{F}_X| + |\mathcal{B}_{\neg X}| + |\mathcal{W}_{\neg X}|))$ (in terms of the lengths of the vectors *before* the updates).

Think carefully about what updates are required. Specifically, which vectors' clauses might have to be modified so as to swap their unwatched literals into watched position? *(circle <u>all</u> that apply)* $\mathcal{U}_X$ $\mathcal{F}_X$ $\mathcal{B}_{\neg X}$ $\mathcal{W}_{\neg X}$ NONE OF THESE

(e) [6 points] zChaff doesn't maintain as much information as the above algorithm. It only maintains two of the above 5 lists (in fact, it doesn't even keep these two lists separate but just maintains their union). Which two?
*(circle <u>two</u>)* $\mathcal{U}_L$ $\mathcal{F}_L$ $\mathcal{B}_L$ $\mathcal{W}_L$ $\mathcal{U}$

(f) [5 points] So why do we need more complicated data structures for WalkSAT than we do for zChaff? Give at least one fundamental reason.[1]

[1] In some ways WalkSAT is also simpler, since it has no unassigned variables.