

Exact Solution

James Tremblay



Problem Description & applications

- Given a collection of 3-literal clauses
- Find the variable assignment that maximizes the number of satisfied clauses

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

- Circuit design – maximizing satisfied constraints
- Scheduling – handle conflicting requirements
- Resource allocation – choosing assignments that satisfy the most conditions.

Inputs / Outputs

n (variables/x's) m (clauses / lines)

Numbers represent which x variable is

Negative numbers represent negative
values

4	5	$(\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge$
-1	-2	$(\neg x_3 \vee \neg x_2 \vee \neg x_4) \wedge$
-3	-2	$(x_4 \vee \neg x_1 \vee x_2) \wedge$
4	-1	$(x_2 \vee \neg x_4 \vee \neg x_3) \wedge$
2	-4	$(\neg x_1 \vee \neg x_2 \vee x_3)$

Solution that satisfies the most clauses

First line is number of clauses satisfied

4	1 F	$x_1 = False$
	2 F	$x_2 = False$
	3 F	$x_3 = False$
	4 F	$x_4 = False$

Polynomial Modification

If the clauses only had 2 variables it would be solvable in linear time

In 2-SAT, implications form a graph with no branching

Just follow arrows until you hit a contradiction

$$(-x_1 \vee -x_2) \wedge (x_2 \vee x_3)$$

In 3-SAT, implications creates a decision tree
with splits and exploring both branches takes time



Reduction, Intro to SAT

Given a collection of literal clauses

$$(\neg x_1 \vee \neg x_2) \wedge$$

Any number of literals per line

$$(x_2 \vee \neg x_4 \vee \neg x_3) \wedge$$

Trying to satisfy every clause

$$(x_5)$$

Outputs: if all clauses are satisfiable or not

Reduction SAT \rightarrow Max 3Sat

No changes to the input

For some number of clauses, k , we ask our SAT solver if it can satisfy at least k clauses.

Use binary search from 0 to m to find the largest k ($\log(m)$)

k is the Max-3SAT solution

Reduction Max 3SAT -> SAT

Clauses with less than 3 literals
add a redundant variable

$$(\neg x_1 \vee x_2) \rightarrow (\neg x_1 \vee x_2 \vee x_2)$$

Clauses with more than 3 literals are broken up into multiple clauses

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \rightarrow (x_1 \vee x_2 \vee b) \wedge (\neg b \vee x_3 \vee x_4)$$

Run Max 3SAT solver and take result x

($x == \#clauses$) answer to SAT

Program

Checks if a clause satisfied

```
def clause_sat(clause, assignment):
    a, b, c = clause
    return [
        (a > 0 and assignment[a-1]) or (a < 0 and not assignment[-a-1]) or
        (b > 0 and assignment[b-1]) or (b < 0 and not assignment[-b-1]) or
        (c > 0 and assignment[c-1]) or (c < 0 and not assignment[-c-1])
    ]
```

Loops over all possible assignments and finds the one that satisfies the most clauses

```
for a in itertools.product([False, True], repeat=n):
    val = sum(clause_sat(c, a) for c in clauses)
    if val > best_val:
        best_val = val
        best_assignment = a
```

Input Changes affect speed

For any ordering the exact solution goes through the same set of assignments and the same number of clause checks

Changing the input order will not reduce the amount of time spent

The only change to input to make program faster is to reduce the amount of variables which reduces the total possible number of assignments

Plot

Exact Runtime vs Input Size n

