

The 3-SAT Decision Problem Exhaustive Search Implementations Team Satisfaction

Christopher Wood, Eitan Romanoff, Ankur Bajoria

April 3, 2013

Agenda

- 1 Problem Statement
- 2 Exhaustive Search Algorithm
- 3 Sequential Program Demo
- 4 Questions

Boolean Satisfiability

Boolean satisfiability is an *NP*-complete decision problem defined as:

$$SAT : \phi \rightarrow \{YES, NO\}$$

Input: Boolean formula ϕ_n on n variables.

$$\phi_5 = (x_1 \vee x_2 \vee \neg x_3) \vee \neg x_4 \wedge x_5 \wedge (\neg x_2 \vee x_3 \wedge (x_4 \wedge x_1))$$

Output: *YES* if there exists a variable truth assignment to the variables in ϕ_n such that it evaluates to true, *NO* otherwise.

$$\phi_n \text{ is satisfiable} \Leftrightarrow SAT(\phi_n) = YES$$

3-SAT \in NP

- A special case of *SAT* that fixes the format of ϕ_n .
- Each input formula is in 3-CNF form:
 - The conjunction (Boolean AND) of arbitrarily many clauses, where each clause is the disjunction (Boolean OR) of exactly three literals (a Boolean variable or its negation).

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

- *SAT* reduces to 3-SAT, so 3-SAT \in NP.

Exhaustive Search for 3-SAT

Input: 3-CNF formula ϕ_n on n variables, **Output:** YES or NO

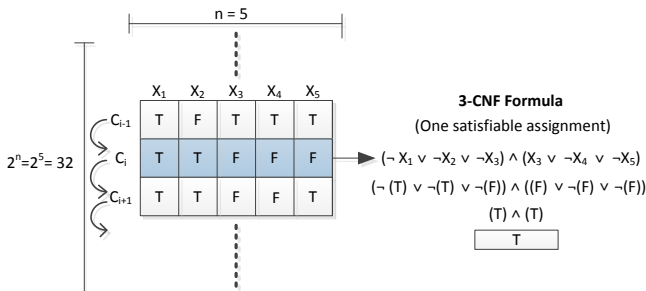
```
1:  $C \leftarrow FALSE^n$  (array of  $n$  False values, the initial configuration)
2: for  $i = 0 \rightarrow 2^n - 1$  do
3:    $SAT \leftarrow TRUE$ 
4:   for all  $clause \in \phi_n$  do
5:     if  $evaluate(clause, C) = FALSE$  then
6:        $SAT \leftarrow FALSE$ 
7:     end if
8:   end for
9:   if  $SAT = TRUE$  then return YES
10:   $C \leftarrow nextConfig(C)$ 
11: end for
12: return NO
```

Exhaustive Search for 3-SAT - A *Very Satisfiable* Example!

Input: $\phi_5 = (\neg X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_3 \vee \neg X_4 \vee \neg X_5)$

Output: Yes

Note: no early termination once a satisfiable solution is found



Evaluating a Clause

- Evaluating a clause depends on how ϕ_n and the variable truth assignments are stored.
 - `boolean[] variables` for variable assignments and `Literal[][3] formula` for ϕ_n .
 - A `Literal` has a variable ID and negated flag

```
for (int c = 0; c < numClauses; c++) {  
    boolean clauseValue = false;  
    for (int l = 0; l < 3 && clauseValue == false; l++) {  
        if (formula[c][l].negated == true && !variables[formula[c][l].id])  
            clauseValue = true;  
        else if (formula[c][l].negated == false && variables[formula[c][l].id])  
            clauseValue = true;  
    }  
    // Check the value of the clause now...  
}
```

The Sequential Solver

Demo time!

$$\phi_5 = (\neg X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_3 \vee \neg X_4 \vee \neg X_5)$$

Questions?

Fire away!