What is SAT?

- Given a propositional formula in CNF, find an assignment to boolean variables that makes the formula true!

  E.g.

  \[ \omega_1 = (x_2 \lor x_3) \]
  \[ \omega_2 = (\neg x_1 \lor \neg x_3) \]
  \[ \omega_3 = (\neg x_2 \lor x_4) \]

  \[ A = \{x_1=0, x_2=1, x_3=0, x_4=1\} \]

  SATisfying assignment!
What is SAT?

- **Solution 1:** Search through all assignments!
  - $n$ variables $\rightarrow 2^n$ possible assignments, explosion!

- SAT is a classic NP-Complete problem, solve SAT and P=NP!

Why SAT?

- Fundamental problem from theoretical point of view
- Numerous applications
  - CAD, VLSI
  - Optimization
  - Model Checking and other type of formal verification
  - AI, planning, automated deduction
Outline

- Terminology
- Basic Backtracking Search
- GRASP
- Pointers to future work

Please interrupt me if anything is not clear!

Terminology

- CNF formula $\varphi$
  - $x_1, \ldots, x_n$: $n$ variables
  - $\omega_1, \ldots, \omega_m$: $m$ clauses

- Assignment $A$
  - Set of $(x, v(x))$ pairs
  - $|A| < n \rightarrow$ partial assignment $\{(x_1, 0), (x_2, 1), (x_3, 1)\}$
  - $|A| = n \rightarrow$ complete assignment $\{(x_1, 0), (x_2, 1), (x_3, 0), (x_4, 1)\}$
  - $\varphi|_A = 0 \rightarrow$ unsatisfying assignment $\{(x_1, 1), (x_3, 1)\}$
  - $\varphi|_A = 1 \rightarrow$ satisfying assignment $\{(x_1, 0), (x_2, 1), (x_4, 1)\}$
Terminology

- Assignment $A$ (contd.)
  - $\varphi|_A = X \rightarrow$ unresolved $\{(x_1,0), (x_2,0), (x_4,1)\}$
- An assignment partitions the clause database into three classes
  - Satisfied, unsatisfied, unresolved
- Free literals: unassigned literals of a clause
- Unit clause: #$\text{free literals} = 1$

Basic Backtracking Search

- Organize the search in the form of a decision tree
  - Each node is an assignment, called decision assignment
  - Depth of the node in the decision tree $\rightarrow$ decision level $\delta(x)$
  - $x = v @ d \rightarrow x$ is assigned to $v$ at decision level $d$
Basic Backtracking Search

- Iterate through:
  1. Make new decision assignments to explore new regions of search space.
  2. Infer implied assignments by a deduction process. May lead to unsatisfied clauses, \textbf{conflict}! The assignment is called conflicting assignment.
  3. Conflicting assignments leads to backtrack to discard the search subspace.

Backtracking Search in Action

\[ \omega_1 = (x_2 \lor x_3) \]
\[ \omega_2 = (\neg x_1 \lor \neg x_4) \]
\[ \omega_3 = (\neg x_2 \lor x_4) \]
\[ x_3 = 0 \Rightarrow x_2 = 0 \Rightarrow x_1 = 0 \]
\[ x_2 = 1 \Rightarrow x_3 = 1 \Rightarrow x_4 = 1 \]

No backtrack in this example!
Backtracking Search in Action

Add a clause

\[ w_1 = (x_2 \lor x_3) \]
\[ w_2 = (\neg x_1 \lor \neg x_4) \]
\[ w_3 = (\neg x_2 \lor x_4) \]
\[ w_4 = (\neg x_1 \lor x_2 \lor \neg x_3) \]

\( x_1 = 0 \Rightarrow x_3 = 1 \)
\( x_2 = 0 \Rightarrow x_3 = 1 \)
\( x_3 = 1 \Rightarrow x_3 = 1 \)
\( x_2 = 0 \Rightarrow x_3 = 1 \)

\{ (x_1,0), (x_2,0), (x_3,1) \}

Davis-Putnam revisited

The fastest known algorithms for deciding propositional satisfiability are based on the Davis-Putnam algorithm. A unit clause is a clause that consists of a single literal.

```plaintext
function Satisfiable (clause list S) returns boolean
  /* unit propagation */
  repeat
    for each unit clause L in S do
      delete from S every clause containing L
      delete L from every clause of S in which it occurs
    end for
    if S is empty then return TRUE
    else if no clause is in S then return FALSE end if
    until no further changes occur end repeat
  /* splitting */
  choose a literal L occurring in S
  if Satisfiable (S \ \ {L}) then return TRUE
  else if Satisfiable (S \ \ {\neg L}) then return TRUE
  else return FALSE end if
end function
```
GRASP

- GRASP is Generalized seaRch Algorithm for the Satisfiability Problem (Silva, Sakallah, '96)

- Features:
  - Implication graphs for BCP and conflict analysis
  - Learning of new clauses
  - Non-chronological backtracking!

GRASP search template

```c
// Global variables:
Clause database

// Return value:
Partial variable assignment

// Auxiliary variables:
Backtracking decision level

GRASP():
{
    return ((Search (L, β) = SUCCESS) ? FAILURE : SUCCESS);
}

// Input argument:
Current decision level

// Output argument:
Backtracking decision level

// Return value:
SUCCESS or FAILURE

// Search (x, A, B):
{
    if (Decide (x) = SUCCESS)
        return SUCCESS;
    while (TRUE):
        if (Reduce (x) = COMFLICT):
            if (Search (x = 1, B) = SUCCESS) return SUCCESS;
        else if (x > 0) { Erase(); return COMFLICT; }
        if (Diagnose (x, B) = COMFLICT) { Erase(); return COMFLICT; }
        Erase();
}
```
GRASP Decision Heuristics

- **Procedure** `decide()`
- Choose the variable that satisfies the most #clauses == max occurences as unit clauses at current decision level
- Other possibilities exist

GRASP Deduction

- Boolean Constraint Propagation using implication graphs
  - E.g. for the clause $\omega = (x \lor \neg y)$, if $y=1$, then we must have $x=1$
  - For a variable $x$ occurring in a clause, assignment 0 to all other literals is called antecedent assignment $A(x)$
    - E.g. for $\omega = (x \lor y \lor \neg z)$,
      - $A(x) = \{(y,0), (z,1)\}$, $A(y) = \{(x,0), (z,1)\}$, $A(z) = \{(x,0), (y,0)\}$
    - Variables directly responsible for forcing the value of $x$
    - Antecedent assignment of a decision variable is empty
Implication Graphs

- Nodes are variable assignments $x = v(x)$ (decision or implied)
- Predecessors of $x$ are antecedent assignments $A(x)$
  - No predecessors for decision assignments!
- Special conflict vertices have $A(\kappa) = \text{assignments to vars in the unsatisfied clause}$
- Decision level for an implied assignment is
  $$\delta(x) = \max \{ \delta(y) | (y, v(y)) \in A(x) \}$$

Example Implication Graph

Current truth assignment: $\{ x_9 = 0@1, x_{10} = 0@3, x_{11} = 0@3, x_{12} = 1@2, x_{13} = 1@2 \}$

Current decision assignment: $\{ x_1 = 1@6 \}$

- $\omega_1 = \overline{x_1} \lor x_2$
- $\omega_2 = \overline{x_1} \lor x_3 \lor x_5$
- $\omega_3 = \overline{x_2} \lor \overline{x_3} \lor x_4$
- $\omega_4 = \overline{x_4} \lor x_5 \lor x_{10}$
- $\omega_5 = \overline{x_4} \lor x_6 \lor x_{11}$
- $\omega_6 = \overline{x_5} \lor x_6$
- $\omega_7 = x_7 \lor x_8 \lor \overline{x_{12}}$
- $\omega_8 = x_7 \lor x_8$
- $\omega_9 = \overline{x_7} \lor \overline{x_8} \lor \overline{x_{11}}$
GRASP Deduction Process

After a conflict arises, analyze the implication graph at current decision level

Add new clauses that would prevent the occurrence of the same conflict in the future ⇒ Learning

Determine decision level to backtrack to, might not be the immediate one ⇒ Non-chronological backtrack
Learning

- Determine the assignment that caused the conflict, negation of this assignment is called conflict induced clause $\omega_C(\kappa)$
  - The conjunct of this assignment is necessary condition for $\kappa$
  - So adding $\omega_C(\kappa)$ will prevent the occurrence of $\kappa$ again

---

Learning

- Find $\omega_C(\kappa)$ by a backward traversal of the IG, find the roots of the IG in the transitive fanin of $\kappa$
- For our example IG,

$$\omega_C(\kappa) = (\neg x_7 \lor x_9 \lor x_{10} \lor x_{11})$$
Learning (some math)

- For any node of an IG $x$, partition $A(x)$ into
  \[ \Lambda(x) = \{(y,v(y)) \in A(x) | d(y) < d(x)\} \]
  \[ \Sigma(x) = \{(y,v(y)) \in A(x) | d(y) = d(x)\} \]

- Conflicting assignment $A_C(\kappa) = \text{causesof}(\kappa)$, where

\[
\text{causesof}(x) = \begin{cases} 
(x,v(x)) & \text{if } A(x) = \phi \\
\Lambda(x) \cup \bigcup_{(y,v(y)) \in \Sigma(x)} \text{causesof}(y) & \text{o/w}
\end{cases}
\]

Learning (some math)

- Deriving conflicting clause $\omega_C(\kappa)$ from the conflicting assignment $A_C(\kappa)$ is straightforward

\[
\omega_C(\kappa) = \sum_{(x,v(x)) \in A_C(\kappa)} x^{v(x)}
\]

- For our IG,
  \[ A_C(\kappa) = \{x_1=0@1, x_2=0@1, x_{10} = 0@3, x_{11} = 0@3\} \]
Learning

- Unique implication points (UIPs) of an IG also provide conflict clauses
- Learning of new clauses increases clause database size
- Heuristically delete clauses based on a user parameter
  - If size of learned clause > parameter, don’t include it

Backtracking

**Failure driven assertions (FDA):**

- If $\omega_C(\kappa)$ involves current decision variable, then after addition, it becomes unit clause, so different assignment for the current variable is immediately tried.
- In our IG, after erasing the assignment at level 6, $\omega_C(\kappa)$ becomes a unit clause $\neg x_i$
- This immediately implies $x_i=0$
Continued IG

Due to $\omega_c(k)$

$9/20/01$  15-398: GRASP and Chaff

Backtracking

Conflict Directed Backtracking

- Now deriving a new IG after setting $x_j=0$ by FDA, we get another conflict $\kappa'$

- Non-chronological backtrack to decision level 3, because backtracking to any level 5, 4 would generate the same conflict $\kappa'$
Conflict Directed Backtracking contd.

\[ A_C(\kappa') = \{ x_9 = 0 @ 1, x_{10} = 0 @ 3, x_{11} = 0 @ 3, x_{12} = 1 @ 2, x_{13} = 1 @ 2 \} \]

\[ \omega_c(\kappa) = (x_9 \lor x_{10} \lor x_{11} \lor \neg x_{12} \lor \neg x_{13}) \]

- Backtrack level is given by

\[ \beta = \max\{ \delta(x) | (x, v(x)) \in A_c(\kappa') \} \]

- \( \beta = d-1 \) chronological backtrack
- \( \beta < d-1 \) non-chronological backtrack

---

**Procedure** Diagnose()

```java
// Global variables:
// Implication graph {
// Clause database \( \phi \)
// Input variable:
// Current decision level \( d \)
// Output variable:
// Backtracking decision level \( \beta \)
// Return value:
// CONFLICT or SUCCESS

Diagnose (d, &beta) {
    \( \omega(e) = \text{Create_Conflict_Induced_Clauses()}; \)  // Using (3.4)
    Update_Clauses Database (\( \omega(e) \));
    \( \beta = \text{Compute_Max_Lower();} \)  // Using (3.7)
    if (\( \beta \neq d \)) {
        add new conflict vertex \( \kappa \) to \( f \); record \( \kappa(e) \); return CONFLICT;
    }
    return SUCCESS;
}
```

9/20/01 15-398: GRASP and Chaff 30
Is that all?

- Huge overhead for constraint propagation
- Better decision heuristics
- Better learning, problem specific
- **Better engineering!**
  
  **Chaff**