Basics of SAT Solving Algorithms

Sol Swords

December 8, 2008
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
What is a SAT problem?

Given a propositional formula (Boolean variables with AND, OR, NOT), is there an assignment to the variables such that the formula evaluates to true?

- NP-complete problem with applications in AI, formal methods
- Input usually given as Conjunctive Normal Form formulas - linear reduction from general propositional formulas
Conjunctive Normal Form

SAT solvers usually take input in CNF: an AND of ORs of literals.

- **Atom** - a propositional variable: $a, b, c$
- **Literal** - an atom or its negation: $a, \bar{a}, b, \bar{b}$
- **Clause** - A disjunction of some literals: $a \lor \bar{b} \lor c$
- **CNF formula** - A conjunction of some clauses: $(a \lor \bar{b} \lor c) \land (\bar{c} \lor \bar{a})$

A formula is *satisfied* by a variable assignment if every clause has at least one literal which is true under that assignment.

A formula is *unsatisfied* by a variable assignment if some clause’s literals are all false under that assignment.
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
The Bare Gist of DPLL-based SAT algorithms

- Perform a depth-first search through the space of possible variable assignments. Stop when a satisfying assignment is found or all possibilities have been tried.

Many optimizations possible:

- Skip branches where no satisfying assignments can occur.
- Order the search to maximize the amount of the search space that can be skipped.
Slightly More Detailed Sketch

Repeat:

- **Decide**: Select some unassigned variable and assign it a value.
  - If all variables are assigned, return SAT.
- **Deduce**: Infer values of other variables that follow from that assignment and detect conflicts.
- **Resolve**: In case of conflict, record a new clause prohibiting that conflict; undo the assignments leading to the conflict.
  - If it’s a top-level conflict (the conflict clause is empty), return UNSAT.
Basic Algorithm

1. Top-level conflict?
   - Yes: Resolve
     - Yes: Conflict?
       - Yes: Add conflict clause, rewind assignments
       - No: SAT
     - No: Decide
       - Yes: Deduce
         - Yes: Resolve
         - No: Top-level conflict?
       - No: Decide
         - Yes: Deduce
         - No: Decide
   - No: Deduce
     - Yes: Resolve
     - No: Top-level conflict?
Ways to make DPLL Faster

- **Decide**: Use a good heuristic to select among unassigned variables
  - activity heuristic based on how often a variable is involved in a conflict
- **Deduice**: Use a good trade-off between speed and completeness
  - Boolean constraint propagation with watched literals
  - Typically about 80% of SAT-solver runtime
- **Resolve**: Take advantage of information revealed by conflicts without over-growing the clause set
  - Learn one or more new clauses at each conflict
  - Backtrack to the "root cause" of the conflict
  - Delete conflict clauses based on an activity heuristic to keep the working set small
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
Two simple rules:

- If all but one of a clause’s literals are assigned FALSE and the remaining literal is unassigned, assign it TRUE.
- If all of a clause’s literals are assigned FALSE, return UNSAT.

Naive algorithm: Inspect each clause and apply the rule; repeat until no new assignments are made.
Motivation for watched literal method

Ideal BCP: Each clause is inspected only after all but one literal is assigned false.
  - Nothing is accomplished by inspecting a clause when it is satisfied or when multiple literals are unassigned.

Best known way to approximate this ideal:
  - Associate each clause with two of its unassigned literals
  - Only examine the clause when one of them is assigned false.
When a literal $a$ is assigned true:

- For each clause $k$ in the watch list of $\overline{a}$, do:
  - If all but one literal $b$ is assigned false, assign $b$ true (and recur);
  - If all literals are assigned false, exit (UNSAT);
  - If any literal is assigned true, continue;
  - Otherwise, add $k$ to the watch list of one of its remaining unassigned literals and remove it from the watch list of $\overline{a}$.

Notes:

- Low overhead, large reduction in number of clause inspections relative to naive algorithms.
- Tricky to maintain all the right invariants so that backtracking doesn’t break the watch lists.
Watched Literals Example

- Watched literals $a, \overline{b}$, all literals unassigned
Watched Literals Example

- Watched literals $a, \overline{b}$, all literals unassigned
- $\overline{c}$ assigned: don’t inspect
Watched Literals Example

- Watched literals \(a, \overline{b}\), all literals unassigned
- \(\overline{c}\) assigned: don’t inspect
- \(b\) assigned: inspect,

\[
\begin{array}{cccc}
a & \overline{b} & c & d \\
\end{array}
\]
Watched Literals Example

- Watched literals $a, \overline{b}$, all literals unassigned
- $\overline{c}$ assigned: don’t inspect
- $b$ assigned: inspect, choose new watched literal $d
Watched Literals Example

- Watched literals $a, \bar{b}$, all literals unassigned
- $\bar{c}$ assigned: don’t inspect
- $b$ assigned: inspect, choose new watched literal $d$
- $\bar{d}$ assigned: inspect,
Watched Literals Example

- Watched literals $a, \overline{b}$, all literals unassigned
- $\overline{c}$ assigned: don’t inspect
- $b$ assigned: inspect, choose new watched literal $d$
- $\overline{d}$ assigned: inspect, propagate $a$

\[ a \overline{b} c d \]
Watched Literals Example

- Watched literals $a, \bar{b}$, all literals unassigned
- $\bar{c}$ assigned: don’t inspect
- $b$ assigned: inspect, choose new watched literal $d$
- $\bar{d}$ assigned: inspect, propagate $a$
- Backtrack to before $b$

\[
a \bar{b} \ c \ d
\]
Watched Literals Example

- Watched literals $a, \overline{b}$, all literals unassigned
- $\overline{c}$ assigned: don’t inspect
- $b$ assigned: inspect, choose new watched literal $d$
- $\overline{d}$ assigned: inspect, propagate $a$
- Backtrack to before $b$
- $a$ assigned: don’t inspect

$a \overline{b} c d$
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
Conflicts can be exploited to reduce the space to be searched.

- Find a conflict (skip the subtree where it’s rooted)
Conflicts can be exploited to reduce the space to be searched.

- Find a conflict (skip the subtree where it’s rooted)
- Analyze the conflict to find a sufficient condition
Conflicts can be exploited to reduce the space to be searched.

- Find a conflict (skip the subtree where it’s rooted)
- Analyze the conflict to find a sufficient condition
- Skip future areas of search space where the condition holds
Conflict Clauses

- To prune branches where $a = \text{false}$, $b = \text{true}$, $c = \text{true}$, add conflict clause $a \bar{b} \bar{c}$.
- Pruning is implicit in BCP.
- Use learned clause to determine how far to backtrack.
  - Backtrack to earliest decision level in which exactly one variable is unassigned.
- How to calculate this clause?
Calculating a conflict clause

To analyze a clause \( c \):

- For each false literal \( x \) in \( c \), either add \( x \) to the conflict clause or analyze the clause \( c' \) which propagated \( \bar{x} \) (heuristic decision.)

Picture: clause \( a \bar{d} \bar{e} \bar{g} \) causes a conflict, yielding conflict clause \( a \bar{b} \bar{c} \).

- Can construct a resolution proof of the new clause from this process:

  \[
  \begin{align*}
  a \bar{d} \bar{e} \bar{g}, \; \bar{g} \bar{d} \bar{c} & \Rightarrow a \bar{d} \bar{e} \bar{c} \\
  a \bar{d} \bar{e} \bar{c}, \; a \bar{h} \bar{d} & \Rightarrow a \bar{e} \bar{c} \bar{h} \\
  a \bar{e} \bar{c} \bar{h}, \; e \bar{h} \bar{c} & \Rightarrow a \bar{c} \bar{h} \\
  a \bar{c} \bar{h}, \; h \bar{b} & \Rightarrow a \bar{b} \bar{c}
  \end{align*}
  \]
Conflict clause heuristics

Issue: Include a literal in the conflict clause, or explore the clause that caused its assignment?

▶ No choice about decision literals
▶ Goal: Small, relevant conflict clauses
  ▶ Possible to generate more than one clause from a conflict, but most solvers don’t
▶ Typical choice is “First UIP” (Unique Implication Point) strategy:
  ▶ Never explore causes of literals assigned due to previous decisions
  ▶ Generate the smallest clause that includes exactly one literal from the current decision level
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
Useful high-level strategic heuristics:

- Choose decision literals by activity heuristic: how often has a literal (recently) been involved in conflicts?
  - Many tweaks possible
  - Choose randomly some small percentage of the time
- Periodically delete some conflict clauses to keep the working set small
  - Various heuristics: activity, size, number of currently-assigned literals
  - A clause is “locked” (may not be deleted) if it is the reason for a current assignment
- Periodically restart the search while keeping some learned clauses
  - Try to avoid “dead ends” where heuristics are pushing in the wrong direction
  - Most solvers increase limitations on backtracks and learned clauses at each restart
Outline

Vocabulary and Preliminaries

Basic Algorithm

Boolean Constraint Propagation

Conflict Analysis

High-level Strategy

Reading
References

- Marques-Silva & Sakallah, *GRASP: A search algorithm for propositional satisfiability*
- Moskewicz et al, *Chaff: Engineering an efficient SAT solver*
- Een & Sorensson, *An extensible SAT-solver*
- Zhang et al, *Efficient conflict driven learning in a Boolean satisfiability solver*