Modern SAT Solvers

CS294 - Automated Deduction
Manu Sridharan

1

Overview

- Focus on techniques used in today's fast solvers
- Introduction to SAT / backtracking search
- Optimizations
- Witness for UNSAT
- Recommended reading
  - Shlyakhter, "Main Techniques for Solving Real-World SAT Instances". http://ilya.cc/area/
  - Zhang and Malik, "Validating SAT Solvers Using an Independent Resolution-Based Checker…"

2

SAT problem

- Input: boolean formula in CNF format
  - Conjunction of clauses, where each clause is a disjunction of literals (AND of ORs)
  - Literal: boolean variable or its negation
- Output: either
  - A satisfying assignment, giving each boolean variable a value 0 or 1 such that all clauses are true
  - UNSAT, indicating no such assignment exists

Formula: $(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_3)$

Satisfying Assignment: $\{< x_1, 1 >, < x_2, 1 >, < x_3, 1 >\}$

3

More on SAT

- Many real-world problems reduce to SAT
  - Software and hardware verification
  - AI planning
- SAT is NP-complete
  - Intractable in the worst case
- But, many large practical instances are tractable
  - Take advantage of higher-level structure

4

Backtracking Search (Davis-Putnam)

- Maintain partial assignment to vars
  - See if it can be extended to a satisfying assignment
- Step 1: boolean constraint propagation (BCP)
  - Assign variables whose setting is implied by current partial assignment
    - Given clause $x_1 \vee x_2 \vee x_3$ and partial assignment { $\langle x_1, 0\rangle$, $\langle x_2, 0\rangle$ }, must have $\langle x_3, 1\rangle$ (clause is *unit clause)*
    - For setting $\langle x_3, 1\rangle$, $x_1 \vee x_2 \vee x_3$ is the *antecedent clause* and { $\langle x_1, 0\rangle$, $\langle x_2, 0\rangle$ } are *antecedent settings*
  - If all variables assigned and no falsified clauses, print satisfying assignment
  - Also called *unit propagation*

5

Backtracking Search, Cont.

- Step 2: search
  - If no clauses falsified by BCP, give some unassigned var a tentative value (a *decision)*
    - Choose variable using *decision heuristic*
    - Mark decision as *open*, since only one setting has been attempted for var
    - Repeat BCP
  - If some clause falsified by BCP (a *conflict)*, backtrack
    - Find last open decision setting
    - Set var from open decision to opposite value, and undo later settings (due to unit propagation)
    - Mark decision as *closed*
    - If no open decisions, formula is unsatisfiable

6

Optimizing BCP

- BCP takes 80-90% of solver time
- Classic implementation:
  - For each clause, keep counts of satisfied, falsified, and unresolved literals
  - When literal is set or unset, visit all clauses with literal and update counters
- Drawback: setting and unsetting literals expensive

7

Watched Literals

- When does a clause matter during search?
  - Going from 2 non-falsified literals to 1 (unit clause)
  - Going from 1 non-falsified literal to 0 (conflict)
- Choose 2 *watched literals* in each clause
  - Invariant: watched literals are non-falsified if clause is not satisfied
  - When literal is falsified, visit all clauses in which it is watched
    - If another unwatched non-falsified literal exists, it becomes watched
    - Otherwise, either unit clause, conflict, or already sat

8

Advantages of Watched Literals

- Fewer clauses visited when literal is set
- Unassignment is constant time!
  - Watched literals unchanged
    - No literals falsified
    - Final watched literals will be first to be unassigned
- Frequent re-assignments of literals faster
  - Once literal is assigned false, becomes unwatched in most clauses
  - So, after unassigning literal, re-assignment is faster
  - Important for certain decision heuristics

9

Learning: Idea

- At conflict, backtracking ensures that current partial assignment not explored again
- But, only a subset of current partial assignment may be responsible for conflict
- Subset can be ruled out by adding a clause
  - To rule out { $\langle x_1, 0\rangle$, $\langle x_2, 0\rangle$ }, add clause $x_1 \vee x_2$
- Learning attempts to find such assignment subsets and add clauses to exclude them

10

Learning: Implementation

- Build *implication graph*
  - Nodes are settings in partial assignment, with special node for conflict clause
  - Edges correspond to antecedent settings
    - From each setting for literals in antecedent clause to implied setting
    - From each setting causing conflict to conflict clause node
- Learned clause is an *antecedent set*
  - Immediate predecessors of conflict clause node an antecedent set
  - Can replace node in antecedent set with all of its predecessors (essentially *resolution*)
  - Try to find small antecedent set (more unit propagation)

11

Learning: Why It Helps

- At a high level, excludes larger parts of search space
  - Extra unit propagation
- As search progresses, new clauses reflect "deeper" conflicts
  - Based on original CNF and previously added clauses
- Independent subproblems are not conflated by decision heuristic

12

Non-Chronological Backtracking

- Normal backtracking flips most recent open decision setting
- Latest decision may not have caused conflict
  - Due to learned clauses
- Non-chronological backtracking flips most recent open decision contributing to conflict
- Easy to implement along with learning
  - For each variable setting, maintain *decision level*, height of decision stack at time of setting
  - Backtrack to latest decision level of literals in learned clause

Restarts

- Solving time varies widely under different decision sequences
  - Some decision sequences much better than average, some much worse
- Restarts simply start the search from scratch after K backtracks
  - May need some randomness in decision heuristic
  - Keep learned clauses to filter search space
  - Increases probability of finding good sequence; hopefully bails out of very bad sequences
- Completeness through gradually increasing K or learning

Decision Heuristic

- How to choose variable to set during search
- Difficult to develop
  - Bad early decision can be very costly
  - Hard to detect badness, esp. with a hard satisfiable problem
- Learning, non-chronological backtracking, and restarts compensate for difficulty

VSIDS

- Variable State Independent Decaying Sum
- For each literal, keep counter of how many learned clauses it appears in
  - Periodically divide by constant to bias to recently learned clauses
- At each decision, choose literal with highest counter
- Partial assignments more likely to lead to solution
  - Learned clauses are resolvents of earlier clauses
  - Assignment satisfying resolvent extendable to one satisfying original clauses
- Possibly leads to shorter learned clauses
  - Learned clauses share more literals, shortening resolvents

Checking UNSAT result

- For satisfiable formulas, satisfying assignment is witness
- Would like "witness" for UNSAT answers as well
  - Optimizations may lead to bugs, eg. in SAT 2002 competition
- CNF formula is UNSAT iff empty clause can be derived using sequence of resolutions
- By instrumenting learning, solver can produce a "trace" showing sequence of resolutions to perform

Producing the UNSAT trace

- Keep all learned clauses
  - Do closed decisions using BCP
- Assign each clause an ID
- For each learned clause, record its ID and ID's of clauses involved in its generation
- Before returning UNSAT
  - Record ID of some final conflicting clause
  - Record all assigned vars, along with values and ID's of antecedent clauses

Verifying the UNSAT trace

- Start with a final conflicting clause
- Repeatedly choose literal from clause, and resolve clause with antecedent of literal's var
  - Check that antecedent is unit clause for var
- Continue until empty clause is derived
- Learned clauses must be reconstructed
  - Construct recursively, using ID's recorded in trace
  - At each resolution step, ensure one var is shared, with opposite polarity
- Original clauses encountered form an *unsat core*
  - Subset of clauses that are still unsatisfiable
  - Useful for explaining unsatisfiability, finding bugs

The End