What have I learned about SAT?

or Tales of an NP-complete but useful problem

Thomas Sewell

NICTA

26th August 2013

Overview

1. SAT
2. Demo of CDCL
3. Modern SAT
   - Competition
   - Pruning
   - Oscillation and Restarts
   - Glue
   - Simplification and Rewriting
4. SAT with proofs
   - Why?
   - RUP & DRUP
   - The future?
5. Why?

CSAT

Survey: what fraction of the audience can name an NP-complete problem.

What fraction can give some evidence?

CSAT

Survey: what fraction of the audience can name an NP-complete problem.

What fraction can give some evidence?

The **C**ircuit **Sat**isfiability problem **CSAT** is: given a directed acyclic circuit (with, say, and/or/not gates), with one output, does there exist a combination of inputs which cause it to output high?

If you beleive that all problems in **NP** can be checked by a circuit connected in a clocked cycle to registers, then **CSAT** is **NP**-complete.

SAT

SAT (also called CNF-SAT) specialises **CSAT** to problems in **CNF**: conjunctive normal form or clausal normal form.

The ciruit is a collection of clauses that are all true (conjoined), and each of which is a disjunction of (possibly negated) variables.

E.g. $x_1 \vee x_2 \vee \neg x_3$, $x_2 \vee \neg x_4$, $x_3 \vee x_4$.

SAT

SAT (also called CNF-SAT) specialises **CSAT** to problems in **CNF**: conjunctive normal form or clausal normal form.

The ciruit is a collection of clauses that are all true (conjoined), and each of which is a disjunction of (possibly negated) variables.

E.g. $x_1 \vee x_2 \vee \neg x_3$, $x_2 \vee \neg x_4$, $x_3 \vee x_4$.

The DPLL algorithm (Davis, Putnam, Logemann, Loveland) is a complete algorithm for deciding satisfiability. It's based on branching and backtracking when necessary.

(time for an aside)

DPLL

In case you didn't get it, the DPLL algorithm:

- Propagate obvious information:
  - $x_1 + (\neg x_1 \vee x_3) \Rightarrow x_3$.
  - $x_1 + (x_1 \vee x_4) \Rightarrow$ nothing.
  - This is called **unit propagation**.
- If you run out of clauses, the unit clauses you have are a satisfying assignment **sat**.
- If you learn the empty clause (from $x_1, \neg x_1$), the problem is unsatisfiable **unsat**.
- If you can't propagate, pick a variable and consider both cases $x_1, \neg x_1$.
  - If sat found on branch, done.
  - If unsat on branch, backtrack to other case.
  - If unsat on both branches, unsat.

What's the big deal about SAT?

Why is SAT a big deal?

- can encode a lot of interesting problems.
- can solve huge problems.

After the discovery of the **CDCL** approach in the 90s, realistic problem sizes shot up from thousands to millions of variables.

That's big enough to reason about CPUs.

CDCL

Conflict **d**riven **c**lause **l**earning improves on backtracking in DPLL. The trick is:

- Remember **why** you know everything.
  - starting clauses.
  - branching choices x_1 or $\neg x_1$.
  - derived clauses (unit propagation).
- If we learn the empty clause, look at its parents.
  - choices were $x_1, \neg x_2, x_{12}$.
  - learn new conflict clause $\neg x_1 \vee x_2 \vee \neg x_{12}$.

This optimisation hugely decreases the cost of backtracking.

Modern SAT

What I've managed to learn about the state of the art:

- **Competitions**: the SAT competition has dozens of entrants in any given year, and the pace of progress is impressive.
- **Fast propagation**: modern SAT solvers are built on fancy clause propagation implementations.
- **Locality**: solvers try to make decisions 'near' previous decisions.
- **Phases**: solvers alternate between sat-focused and unsat-focused phases. Phase-saving and rapid restarts are apparently important.
- **Pruning**: clause propagation time grows with the clause database. Pruning the database speeds things up.
- **Glue**: pruning to "glue" clauses, which have a linear blocks distance of 2, works well. Whatever that means.
- **Rewriting**: preprocessing the problem into an equisatisfiable problem. Valuable especially as a first step.

WARNING: this may be wildly inaccurate.

SAT with Proofs

So that is all pretty useful but . . . do you trust it?

SAT with Proofs

So that is all pretty useful but . . . do you trust it?

The SAT competition comes with a proofs category. Solvers augment unsat judgements with some kind of guidance for a checker.

Some solvers produce full resolution proofs.

The RUP format (reverse unit propagation) of a proof is a series of clauses that can be learned by unit propagation only. The conflict clauses of a CDCL solver in the order they are learned form a RUP proof.

The DRUP format adds clause deletion, to speed up unit propagation.

SAT with Rewriting and Proofs

What if we want both rewriting and proofs?

The previous proof formats all assume the solver and checker share the problem representation and strategy. If the problem has been rewritten, that isn't so true any more.

SAT with Rewriting and Proofs

What if we want both rewriting and proofs?

The previous proof formats all assume the solver and checker share the problem representation and strategy. If the problem has been rewritten, that isn't so true any more.

What if we put more steps between the checker and solver?

Why

Why have I subjected you all to this rambling?

I have some SMT proofs I'd dearly like to replay in Isabelle/HOL or HOL4. This can be thought of as a generalisation of replaying SAT with rewriting only:

- the rewriting step is bigger and more complicated.
- some facts come out of nowhere (the SMT theory engine).
- the final checker is really really slow.

Why

Why have I subjected you all to this rambling?

I have some SMT proofs I'd dearly like to replay in Isabelle/HOL or HOL4. This can be thought of as a generalisation of replaying SAT with rewriting only:

- the rewriting step is bigger and more complicated.
- some facts come out of nowhere (the SMT theory engine).
- the final checker is really really slow.

That is all.