Lecture 8 SAT algs Sep 29

Top-Down algs for SAT/UNSAT

Four Central Heuristics

(1) **Decision scheme** – Choose the next variable to branch on, what assignment?

(2) **Unit Propagation scheme** – Detects **units** (variables that must be set a particular way) and sets them.

$F = x_1 \wedge (\bar{x}_1 \vee x_2)$

(3) **Clause Learning Scheme** – From **conflicts** (i.e. when partial assign. falsifies a clause) generate new clauses.

(4) **Restart Scheme** – Decide to throw out the current partial assignment, keep learned clauses.

Algorithmic Schema [Atserias-Fichte-Thurley 11]

Throughout we maintain a **state** S, which is an ordered partial assignment to the variables of F.

We also maintain a set of clauses, C, initially $C = F$.

MAIN If S satisfies all of C, then stop output S
If $C|_S$ contains $\perp$ then goto CONFLICT
If we can unit propagate, goto UNIT
Else goto DECISION

CONFLICT Apply Clause Learning scheme to learn clause C, add to $\mathcal{C}$
Check Restart Scheme, if we restart now set $S = \emptyset$ goto MAIN
Else, remove assignments from S until $C \wedge S \neq \bot$, goto MAIN

UNIT Apply Unit Prop. Scheme and repeatedly set units until none remain — goto MAIN

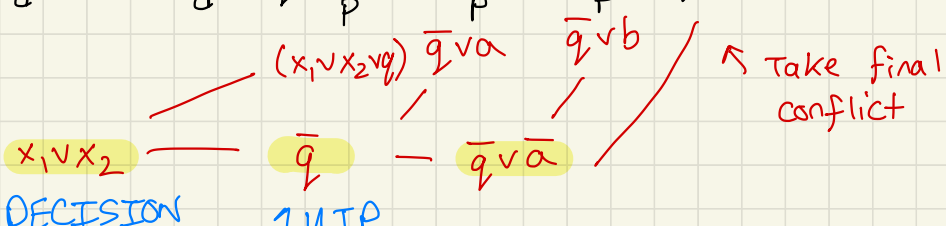
DECISION Apply Decision Scheme to get a new assignment $X :=_d b$ to be added to S, goto MAIN.

Clause Learning?

(A simplified explanation)

ex) $F = (x_1 \vee x_2 \vee q) \wedge (\bar{q} \vee a) \wedge (\bar{q} \vee b) \wedge (\bar{q} \vee \bar{a} \vee \bar{b})$

$x_1 \overset{d=1}{\underset{d}{0}}$ $x_2 \overset{d=2}{\underset{d}{0}}$ $q \overset{d=2}{\underset{p}{1}}$ $a \overset{d=2}{\underset{p}{1}}$ $b \overset{d=2}{\underset{p}{1}}$ $\bar{q} \vee \bar{a} \vee \bar{b}$



Two Heuristics

DECISION := Resolves all possible literals from unit props away. Result is a subset of decision literals!

If S is a state then the **decision level** of an assignment $x=b$ in S is the number of decisions made in S up to (including) $x=b$.

A clause is **asserting** w.r.t. S if there is exactly one literal of the highest decision level.

1UIP := Resolve backwards until we get the first asserting clause.

First unique implication point

	Theoretical	Algorithm Schema Implementations	Practical
	PPSZ	PREPROCESSING →	CDCL Algorithms
Decision :=	Uniformly at random (variable and assignment)		VSIDS heuristic
Unit :=	D-implication (looks at a set of D clauses, check if any unit is implied).		standard (only fix unit clauses)
CL :=	None		1UIP
Restart :=	Restarts every time		Really varies from solver to solver
	↑ We can actually analyze run-time theoretically $2^{(1-\frac{o(1)}{k})n}$		↑ No good theoretical analysis of run-time

CDCL as a proof system

Sep 29

If F is unsatisfiable, then running one of these algs on F essentially outputs a resolution proof!

∴ Algs will take exponential time on

PHP, Tseitin on expanders, Random formulas, ...

just because of resolution lower bounds!

Question If there is a short resolution proof, can these algorithms find it?

⟹ NO (unless P = NP) (automatizability!)

Question 2 What if we allow some "limited" non-determinism in some heuristics?

↗

CDCL "becomes" a propositional proof system!

Thm [Beame-Kautz-Sabharwal 04 ??]

CDCL can efficiently simulate resolution, with non-deterministic

- Decisions
- Unit Propagations

and also use **any** clause-learning scheme that is non-redundant, along with restarting every time.

Thm 2 [BKS 04]

COCL efficiently simulates resolution with non-det

1. Pre-processing
2. variable branching

and a fixed CL scheme "FirstNewCut", greedy unit propagation, and no restarts

Thm 3 [PD 09, AFT 11] (Absorption-theoretical)

efficient simulation is possible with non-det.

1. Variable branching

Decision, 1UIP

and greedy unit prop, any asserting CL scheme, and where we restart after every conflict.

Open Question Understand the power of restarts!

e.g. Can you prove Thm 3 without restarts?

Open Question Try to prove a good theoretical upper bound on the run-time of COCL.

↕

Open Question Understand COCL operates on $n^{O(\sqrt{n} \log s)}$ satisfiable inputs

Other SAT algs

- Schöning's Algorithm for k-SAT [Sch 01]
  (local search algorithm)
- Dynamic programming (no ref. off top of my head)
- Survey propagation (connections to random k-SAT, theoretical physics)