Optimization Methods CS508

May 30, 2020

SAT Solver : DPLL with heuristics

Submitted by :

Vandana Mishra

194101055

SAT

SAT is a decision problem. Given a propositional formula , is it satisfiable or not. The brute force method to solve this problem is to check all possibilities, i.e. assignments to the variables and check if it is satisfiable or not. This method is known as the **method of truth table** and computationally expensive process. As for n variables formula, there are 2n possibilities to check. Hence **SAT is NP-Complete.**

SAT Solver

Tool to solve SAT is called the SAT solver. Solvers can be implemented with any of below mentioned algorithms:

- Algorithms that are based on existential quantification
- Algorithms based on inference rules
- Algorithms based on search
- Algorithms based on the combination of search and inference

Approach Taken : DPLL with heuristics

DPLL :

DPLL is an optimization of DP algorithm for SAT solvers. DPLL is based on **algorithms based on search.**

Purpose for choosing this algorithm is that DPLL forms the basis of current more efficient SAT solvers.

Basic idea of DPLL :

- First apply unit resolution as long as possible.
- If you can't proceed by unit resolution or trivial observations then choose a variable say 'p' , introduce the cases p and ¬p, and for both cases go on recursively.

This is the algo which is implemented, below is the algorithm followed by detailed explanation of the sub points:

Backtrack-search algorithm of Davis, Putnam, Loveland and Logemann (DPLL)

dpll-(CNF Δ): returns a set of literals or unsatisfiable. Variables are named P1, P2, ...

1. CNF = unit-resol(CNF)
2. if Δ = {} then
   2.1. return {}
3. else if {} ∈ Δ then
   3.1. return unsatisfiable
4. else if L =dpll-(BCP(Δ|P)) = unsatisfiable then
   4.1. return L ∪ {P}
5. else if L =dpll-(BCP(Δ|¬P)) = unsatisfiable then
   5.1. return L ∪ {¬P}
6. else
   6.1. return unsatisfiable

Unit resolution technique : unit-resol(CNF)

This is quite a simple technique, which suggests that the unit clauses in the cnf can be used to reduce the formula, i.e. after application of unit-resol , the no. of clauses in CNF definitely decreases.

Steps of unit resolution:

Pick a unit clause, apply conditioning over CNF based on the picked up literal. Here BCP denotest that conditioning. For the literal say p (unit clause)

- Remove all the clauses from CNF which contain literal p. As these clauses will always be true, irrespective of other literal assignments.
- Remove -p from clauses which contain -p. As -p is false , hence it doesn't contribute to the satisfiability of that clause.

After the unit resol and BCP , CNF must be decreased in size. Hence termination is guaranteed. All the unit clauses present initially in the CNF formula will contribute to the assignment.

Function unit-resol, which applies to a CNF Δ and returns two results:

- I: a set of literals that were either present as unit clauses in Δ, or were derived from Δ by unit resolution.
- Γ: a new CNF which results from conditioning(BCP) of Δ on literals I.

When all the unit clauses exhausted, then a variable will be decided from the current CNF formula, and again applying dpll on CNF after applying BCP on CNF with this chosen literal. If CNF is not satisfied then it will backtrack and set variable to negation of variable, and follow this procedure recursively.

Now, here is one catch:

Efficiency of the presented algorithm strongly depends on the choice of the variable. For which may heuristics are there.

Following heuristics are implemented :

- MO : Most_often (variable will be chosen which has maximum appearances)
- SPC : Shortest_positive_clause (as the name suggest, variable will be the first literal of the shortest positive clause)
- JW : Jeroslow_wang (count the appearances and provide the weight to the literal based on the length of the clause it appeared.)

Results:

	Execution time based on heuristics(in sec)			
INPUT FILE	**MO**	**SPC**	**JW**	**Is SAT ?**
aim-50-1_6-yes1-4.cnf (50 vars, 80 clauses)	0.00800	0.01301	0.01401	Yes
quinn.cnf (16 vars, 18 clauses)	0.00200	0.00200	0.00200	Yes
unsat3.cnf (13 var, 34 clauses)	0.000999	0.00100	0.001000	No
aim-100-1_6-no-1.cnf (100 vars, 160 clauses)	20.002	20.4003	21.0234	No

Disadvantages:

- Slow, dpll even with applied heuristics is very slow for CNFs, where number of variables are greater than or equal to 100**.**
- **Reason :** Chronological backtracking.

Solution :

Backjump instead of chronological backtrack, in addition to the learning from the conflicts occurred. So while backtrack , the same mistakes can't get repeated. Hence if conflict is detected , then a clause learned from that conflict needs to be added to the CNF. So that early decision can be possible.

This approach or algo is known as CDCL , conflict driven clause learning and based on the algorithm of searching and inference. (tried to implement this with the help of implication and eduset.cpp code, but it was not working , hence sending the working program based on dpll algorithm).