SAT SOLVER

CS-508 Optimization Methods - 160101061 & Saurabh Bazari

How to run code

Python file (satsolver.py)

- Run command: python3 satsolver.py <path_of_cnf_file>.
- Input format should be in the standard DIMACS CNF.
- Output format has time-spent, Number of Learned Clauses, Decision Count, Implication Count, (SAT or UNSAT).
- If SAT then the assignment of literal

Python Juptyer notebook file (SatSolver.ipynb)

- An input file in the same folder where SatSolver,ipynb is present. The input file name must be "input.cnf"
- Run all cell
- Output format has time-spent, Number of Learned Clauses, Decision Count, Implication Count, (SAT or UNSAT).
- If SAT then the assignment of literal.

Data Structure

1. "LitCalMatrix" is the dictionary in which we **map each literal to indices of clauses** in which that literal is present.
2. "Formula" is the list in which we **store all clauses**. All clauses are the list of literals. All literals are represented as an integer.
3. "litValues" is the list of integer which contain the **value of literals**. Values of literal only -1, 0, 1. -1 means literal is unknown, 0 means literal is false and 1 means literal is true.
4. We use "unknown" as a global variable with value -1.
5. "level" is an integer that maintains the **number of manually decided literals**.
6. "levelAllLits" is a dictionary that maps each level to a list of literals. For each level, the mapped list stores the **literals which are assigned by the BCP** method.
7. "adjMatrix" is a list of nodes, represent the adjacency list of a graph. There is a node for each literal. Whenever we assign a value to any literal, we create a node of that literal and store information like clause index (from which clause current literal assigned), the value of literal and current level. We can find the parent of the current node by using the clause index. For the root node of each level, there is no parent so we assign a clause index of the root node is None.

Implementation

I implement the CDCL with some optimization Techniques. Here, I explain some basic flow of my code.

1. Read Input
    a. Open the input file and read the file line by line.
    b. If the line starts with ‘c’ then that is a comment. So ignore that line.
    c. If the line starts with ‘p’ then that contains the count of clauses and the count of literals.
    d. Else that line contains the clause as a set of integers ending with 0.
2. Update literal values, literal to clauses mapping and adjacency list of graph
    a. Set unknown value for each literal
    b. Set Null node for each literal
    c. For each literal store the indices of clauses where that literal is present.
3. Check if there is any pure literal is present or not (**Optimization**)
    a. If any literal is present in cnf formula and negation of literal in not present then we assign literal is true and create a node in the graph with current level 0.
4. Main Algorithm -> Till all clauses are not true we repeatedly run this algorithm.
    a. We find that unknown valued clauses in which only one unknown literal, all other literals are false. So that unknown literal must be true. From this, maybe we get some more unit literals from assigned unit literals in the same level. We recursively find the unit literals and assign them. FInally we not no more unit literals or conflict on our decision.
    b. If there is no conflict we make one more decision and again repeat the BCP
    c. If there is some conflict, we have to learn a new clause so, we cannot get the same conflict. A new clause and a new level get as output from the Conflict Analysis function which is implemented as a standard Conflict-Driven Clause Learning Algorithm. We have to set our current level to a new level so all intermediate level assignments, decisions, graph nodes are removed.
5. If we get an assignment with all clauses true then the SAT or we get a conflict in level 0 or a new level is negative then UNSAT.

Optimization Techniques:

1. **Pure Literal** - Check if there is any pure literal is present or not. If any literal is present in cnf formula and negation of literal in not present then we assign literal is true because, for each clause, a literal value will be positive or still unknown not false.

2. **Decision Heuristic** - Instead of picking randomly unknown value literal, we apply some technique for that. We choose that unknown literal which has more occurrence in cnf formula and learned clauses.
3. **BCP Iteration** - For recursively BCP interaction on all clauses. For the first time, we apply iteration on all clauses and we get sone unit clauses. But from the second time, the only unit Clause possibility in which negation of recently unit literals are present. So instead of iteration on all clauses, now we iterate only on some number of clauses.
4. **Update Input -** If any clause has both literal and negation of literal. So we remove that clause. Because that clause is always true.
5. **BackPropagation** - Whenever a conflict occurs instead of change value of the last decided literal. We analyze the conflict and find the new level at that point all literal values are correctly decided. For this we using CDCL Algorithm implementation.

Results

Input File Name	Time	New Clauses Count	Decision Count	Implication Count	Result
UNSAT					
unsat	0.0001 sec	1	1	5	UNSAT
unsat1	0.0003 sec	3	3	11	UNSAT
unsat2	0.0001 sec	0	0	3	UNSAT
unsat3	0.002 sec	11	15	80	UNSAT
SAT					
input	0.0001 sec	0	1	1	SAT
input1	0.05 sec	16	49	584	SAT
input2	0.0004 sec	0	7	2	SAT
bmc-2	6.6 sec	53	389	13828	SAT
bmc-3	161 sec	160	659	170850	SAT
bmc-5	85.72 sec	89	1348	42607	SAT
bmc-7	68.79 sec	54	967	35289	SAT