End Semester Assignment

Dhananjay Shukla 194101018
CS508 - Optimization Techniques
IIT Guwahati

June 12, 2020

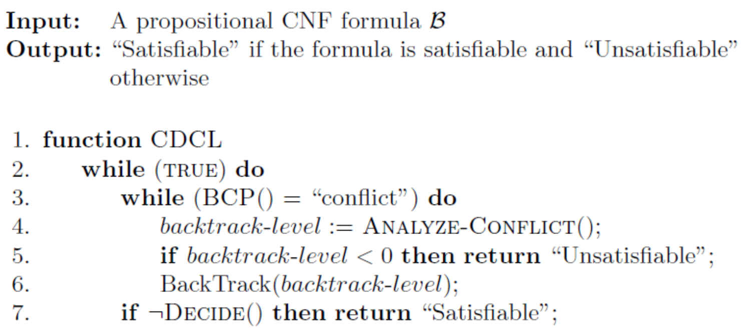
Conflict Driven Clause Learning (CDCL)

CDCL is an algorithm for solving the Boolean satisfiability problem (SAT). CDCL algorithm was developed overtime as a series of improvement to the DPLL framework. CDCL framework is based on traversing and backtracking on a directed cyclic graph, learning new clauses in the process.

CDCL Framework

CDCL solver progresses by making a decision about a variable and its value, propagating implications of this decision and backtracking in case of conflict i.e. checks if any clause evaluates to false under current assignment of variables. Every decision made is associated with a decision level and the assignments implied by a decision are associated with its decision level. Unary clauses (clauses with only a single literal) are associated with decision level 0 as ther are self implied.

Algorithm



Input: A propositional CNF formula $\mathcal{B}$
Output: "Satisfiable" if the formula is satisfiable and "Unsatisfiable" otherwise

```
function CDCL
    while (TRUE) do
        while (BCP() = "conflict") do
            backtrack-level := ANALYZE-CONFLICT();
            if backtrack-level < 0 then return "Unsatisfiable";
            BackTrack(backtrack-level);
        if ¬DECIDE() then return "Satisfiable";
```

Figure 1: CDCL-SAT Algorithm from [1].

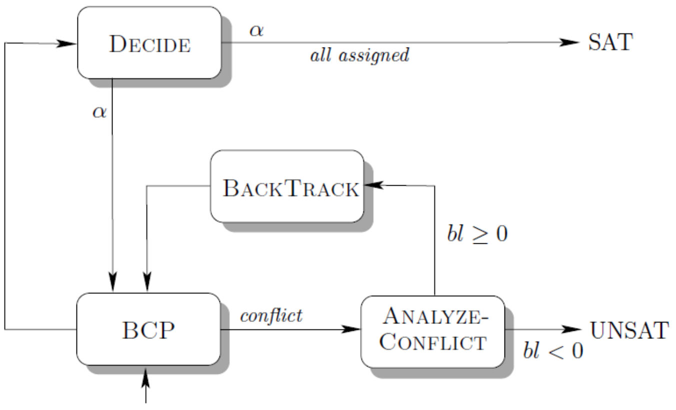


Figure 2: High Level Overview of CDCL Algorithm from [1].

Main Components:
1. Decide() : This function checks whether all variables are assigned or not and if not assigned then chose a variable based on some heuristic for assigning a truth value. Returns **FALSE** if and only if there are no more variables to assign else returns **TRUE**.
I have used two heuristics to chose a variable for assigning:

- **Ordered :** It is a naive approach in which we chose a variable sequentially i.e. the first unassigned variable from the list of unassigned variables.
- **Dynamic Largest Individual Sum(DLIS):** For a given variable ,$C_{x,p}$ is the number of unresolved clauses in which x appears positively, and $C_{x,n}$ is the number of unresolved clauses in which x appears negatively. Then we find 2 literals a, and , b such that $C_{a,p}$ is maximum, and $C_{b,n}$ is maximum. If, $C_{a,p} > C_{b,n}$ then we choose a to be **TRUE**, otherwise b to be **FALSE**.
- "DLIS" performs better i.e. takes less time when compared with "ordered".

Heuristic	unsat1.cnf	sat1.cnf
Ordered	0.54s	0.13s
DLIS	0.47s	0.06s

Table 1: Comparison table for Decision Heuristic for test files unsat1.cnf and sat1.cnf

2. BCP(): It repeatedly uses **Unit Clause Rule** through which implication of the decision currently made are propagated until a comflict is encountered. It finds a unit clause (clause with all literals but one assigned value **False** and one is unassigned) and assigns the unit literal such that unit clause evaluates to **TRUE**. This function uses a DAG i.e. Implication Graph for constraint propagation.

Input : Takes no input.
Output : Returns Conflicting clause if conflict is encountered else return **None** if there are no more implications.

3. Analyze-Conflict(): This function is responsible for computing the backtrack level and adding new constraints on search in the form of new clauses.This function plays a major role in detecting global unsatisfiability.
Input : Conflicting Clause returned by BCP() function.
Output : Returns **-1 and None** if conflict is at level 0 else decision level to which solver should backtrack to and also learned clauses.

- **Unique Implication Point (UIP):** A unique implication point (UIP) is any node at the current decision level such that any path from the decision variable to the conflict node must pass through it.
- Analyze-Conflict() function traverses the graph to find a UIP to stop search and then new clause is learned through this UIP of conflict clause.

4. Backtrack(): Sets the current decision level dl (returned by Analyze-Conflict() function) and erases assignments at decision levels larger than dl.

Implementation and Results

To implement CDCL algorithm i have used two classes and one driver function namely:

- **Class CDCL:** This class makes the heart of the CDCL algorithm defining all of the above major functions as mentioned above.
- **Class NODE:** This class represents the node of implication graph used in CDCL implementation.
- **Driver Function "main":** It calls **solve()** method of CDCL class by creating its object and prints result of satisfiability for the test file and time taken etc. on the stdout. It is the point where the input file is specified.

Benchmark
Six test files in DIMACS CNF format has been used to test the CDCL implementation.

- **unsat1.cnf :** 100 Variables, 160 Clauses - Unsatisfiable
- **unsat2.cnf :** 5 Variables, 9 Clauses - Unsatisfiable
- **unsat2.cnf :** 13 Variables, 34 Clauses - Unsatisfiable
- **sat1.cnf :** 50 Variables, 80 Clauses - Satisfiable
- **sat2.cnf :** 20 Variables, 91 Clauses - Satisfiable
- **sat3.cnf :** 20 Variables, 91 Clauses - Satisfiable

References

[1] Daniel Kroening and Ofer Strichman. *Decision procedures*. Springer, 2016.