CS202 Assignment 2 Report
SAT Solver

Implementation:

We have used the **DPLL (Davis–Putnam–Logemann–Loveland) algorithm** to implement the SAT solver. The reason is that most modern SAT Solvers are based on the DPLL algorithm. DPLL is a backtracking based algorithm, in which, if at any point we find that the formula can't be satisfied then we backtrack, and until we have checked all the possible interpretations we keep on going. The following heuristics are used in DPLL:

1. **Unit Propagation**: This refers to removing the singular clauses (clauses containing a single literal, eg. {p}). We can remove them from the formula since the only way the formula can be satisfied is that they have to be true. We can also remove the negation of the literal in all the clauses (where it occurs), since the negation of the literal would evaluate to false.
2. **Pure Literal Elimination**: If in a formula, a particular literal (say p) occurs only as p or only as ~p, then we can assign p (or ~p) as true. So all the clauses where p (or ~p) occur will become true and we can remove them from the formula.

Basic Pseudo-Code for DPLL algorithm

Input: Formula ϕ (set of clauses)
Output: True or false (depending on whether the formula is SAT or UNSAT respectively)

```
DPLL(Formula ϕ)
{
        while (ϕ has unit clause)
                ϕ:= unit_propagation(ϕ)
        while (ϕ has pure literal)
                ϕ:= remove_pure_literal(ϕ)
        if (ϕ is empty)
                return True
        if(any clause of ϕ is empty)
                return False
        assume a literal l present in the formula
        return DPLL(ϕ/\{l}) || DPLL(ϕ/\{~l})

}
```

We have also used one more heuristic in order to increase the efficiency of the SAT solver. At every step of the DPLL algorithm, where we are supposed to assume a literal, we assume the literal whose **occurrence (in both forms, p and ~p) is maximum**. This would help us simplify the formula a lot more in the next step.

Assumptions

- Input is in the DIMACS format
- comments (lines starting with letter c) are allowed only at the starting

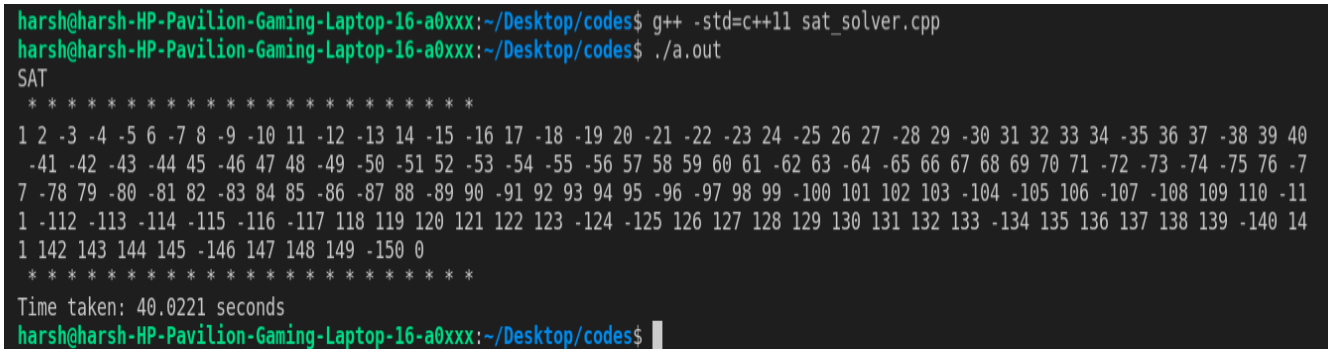
Limitations

- At present, the SAT solver is not very fast when the input has a large number of variables or clauses (of the range > 200) and for some cases the case for n=150 takes 1.5 to 3 minutes.
- The CDCL (Conflict-Driven Clause Learning Algorithm) hasn't been used in our Solver, using it would have improved the efficiency
- The Solver can give only one satisfying model if the formula is SAT, it can't find the number of models nor can it find all the models.

References:

1. https://en.wikipedia.org/wiki/DPLL_algorithm
2. https://www.youtube.com/watch?v=xFpndTg7ZqA

Sample Input and Output



Time taken for the given benchmarks

	A	B	C	D	E	F
1	**File name**	**Time (in s)**	**File name**	**Time (in s)**	**File name**	**Time (in s)**
2	uf20-01.cnf	0.123	uf150-01.cnf	40	uuf150-01.cnf	92
3	uf20-02.cnf	0.002	uf150-02.cnf	33	uuf150-02.cnf	133
4	uf20-03.cnf	0.002	uf150-03.cnf	30	uuf150-03.cnf	115
5	uf20-04.cnf	0.002	uf150-04.cnf	5	uuf150-04.cnf	95
6	uf20-05.cnf	0.001	uf150-05.cnf	0.2	uuf150-05.cnf	92
7	uf20-06.cnf	0.003	uf150-06.cnf	14	uuf150-06.cnf	84
8	uf20-07.cnf	0.002	uf150-07.cnf	43	uuf150-07.cnf	77
9	uf20-08.cnf	0.003	uf150-08.cnf	1.44	uuf150-08.cnf	73.5
10	uf20-09.cnf	0.002	uf150-09.cnf	6.6	uuf150-09.cnf	73
11	uf20-010.cnf	0.002	uf150-010.cnf	13.23	uuf150-010.cnf	131
12	uf20-011.cnf	0.003	uf150-011.cnf	1.1	uuf150-011.cnf	132
13	uf20-012.cnf	0.002	uf150-012.cnf	150	uuf150-012.cnf	109
14	uf20-013.cnf	0.007	uf150-013.cnf	5	uuf150-013.cnf	87
15	uf20-014.cnf	0.002	uf150-014.cnf	60	uuf150-014.cnf	230
16	uf20-015.cnf	0.055	uf150-015.cnf	20	uuf150-015.cnf	181
17					uuf150-016.cnf	65
18					uuf150-017.cnf	105
19					uuf150-018.cnf	615
20					uuf150-019.cnf	96
21					uuf150-020.cnf	142
22						
23						
24						