Assignment 2: Implementing a SAT Solver

Vidish Chandekar (200291)
Ujjwal Kumar (201059)

Implementation:-

1. We've used the Davis-Putnam-Longmann-Loveland (DPLL) to find whether the satisfiability of the given CNF clauses (Formula).
2. It is a complete, back-tracking based search algorithm.
3. The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the *splitting rule*, as it splits the problem into two simpler sub-problems. The simplification step essentially removes all clauses that become true under the assignment from the formula, and all literals that become false from the remaining clauses.
4. DPLL is special because it enhances the back tracking algorithm with the following steps:-
    a. **Unit Propagation:** If a clause is a *unit clause*, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. Unit propagation consists in removing every clause containing a unit clause's literal and in discarding the complement of a unit clause's literal from every clause containing that complement.
    b. Pure Literal Elimination: If a propositional occurs with only one polarity in the formula, it is called *pure*. A pure literal can always be assigned in a way that makes all clauses containing it true. Thus, when it is assigned such a way, these clauses do not constrain the search anymore, and can be deleted.
5. Unsatisfiability of a given partial assignment is detected if one clause becomes empty, i.e. if all its variables have been assigned in a way that makes the corresponding literals false. Satisfiability of the formula is detected either when all variables are assigned without generating the empty clause, or, in modern implementations, if all

clauses are satisfied. Unsatisfiability of the complete formula can only be detected after exhaustive search.

6. This SAT Solver was able to solve test cases up to 20 variables (91 clauses) but was taking too much time when trying to solve for 150 literals (645 clauses). Hence an even better refinement was necessary.
7. In order to refine even further we implemented Heuristics to get a greedy algorithm. So rather than choosing and assigning a random variable we used specific formulae to assign literals so that the algorithm makes more logical choices when it comes to choosing literals. The Two Heusterics we used are:-
    a. **Jeroslow Wang Heuristic:** The goal is to assign variables with high occurrences and short clauses.
    Choose the literal x with a maximum J(x).

$$J(x) = \sum 2\text{^}(-|c|) \quad [x \in c, c \in F]$$

    Here, x is a literal in clause c which is in Formula F and J(x) is a weighting system.
    We assigned variables with high occurrences and short clauses so that it is easier for assignment, unit propagation and back tracking.
    b. **Most Often:** Rather than using fancy formulas we can also, at every stage of the algorithm, assign the literal which has the maximum occurrence. By assigning maximum occurrence we are able to eliminate more clauses and literals at every stage of the execution.

Assumptions: Aside from the fact that the given input is in CNF form in a .cnf file, there aren't any assumptions.
Limitations: The algorithm might take a couple of extra seconds to execute larger CNF Formulae.

ReadMe: There would be 3 .py files

1. **dpll.py:** Uses random choice assignment. Was able to Solve for 20 variables (91 clauses) but not for 150 variables (645 clauses).
2. **dpllJW.py:** Uses Jeroslow Wang Formula for assignment. Faster and is able to solve all the test cases.
3. **dpllMO.py:** Assignment of maximum occurring literal.