SAT Solver

Approach:

The SAT solver uses the classic DPLL algorithm. The algorithm follows the general procedure:

1) Make an assumption on a literal
2) Apply unit-propagation and pure-literal continuously till the formula does not change
3) Repeat

Whenever there is a contradiction, backtracking is done till the appropriate decision level.

Implementation:

The SAT solver was implemented in Java. Apart from the main program, an auxiliary **Clause** class has also been defined with suitable data members and functions to enhance the readability of the program.

The unoptimized algorithm to solve the formula by taking a decision at each level takes an intractable amount of time. Hence, the key idea lies in the usage of unit-propagation optimization, pure-literal optimization and a heuristic for the next literal to assume.

Unit Propagation: For clauses with only one unresolved variable, immediately assign the remaining literal with the forced value

Pure Literal: For a literal with only one polarity among all the unsatisfied clauses, immediately assign that polarity to the literal

Heuristic: For the choice on the next literal to assume, choose a literal among an unsatisfied clause with the MINIMUM size

The recursion used in the program does **NOT COPY** over the formula to the functions, making our SAT solver space efficient..

Miscellaneous:

i) If the file is given in DNF format then the algorithm will print a “model” that renders the DNF UNSAT.

ii) We’ve also implemented a SAT checker in Java which checks the model printed by the algorithm with the clauses.

Assumptions:

i) We assumed that the file will be in CNF format with appropriate comments.

ii) We assumed that no comments arise until all the clauses end.

Limitations:

i) The solver is slow for a higher number of literals. Some UNSAT formulas containing 150 literals can take as much as 40 seconds.