CS154 Project

Niraj Mahajan (180050069)
Tanmay Mandal (180050109)
Raaghav Raaj (180050082)

The SAT Conundrum

Introduction

In this project, we have implemented a functioning Satisfiability Solver (CNF SAT solver), which takes a boolean formula as its input and returns a solution , if it exists, or else, false, which is based on a heuristic approach.

We have further used this Satisfiability Solver to Demonstrate how Boolean problems, like a Sudoku, or N-Queens can be solved along with another application of SAT solving in scheduling of tournament matches.

Program Design

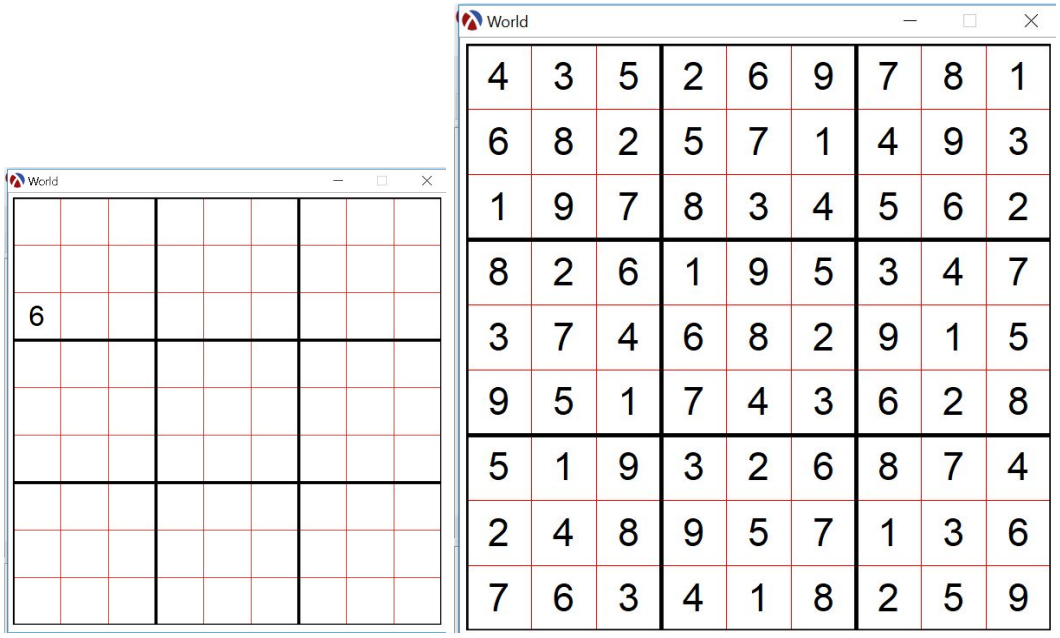
1. **SAT Solver:**

   a. Our SAT Solver is based on the DPLL algorithm. But since this algorithm solely relies on back-tracking, to improve the efficiency, we have also used heuristics like unit-propagation, check for pure-literals, check for binary clauses. We are converting any boolean formula input for the SAT Solver into Conjunctive Normal Form (CNF Format). A conjunctive normal form consists of 'boolean and' operation on clauses which themselves contain only 'boolean or'

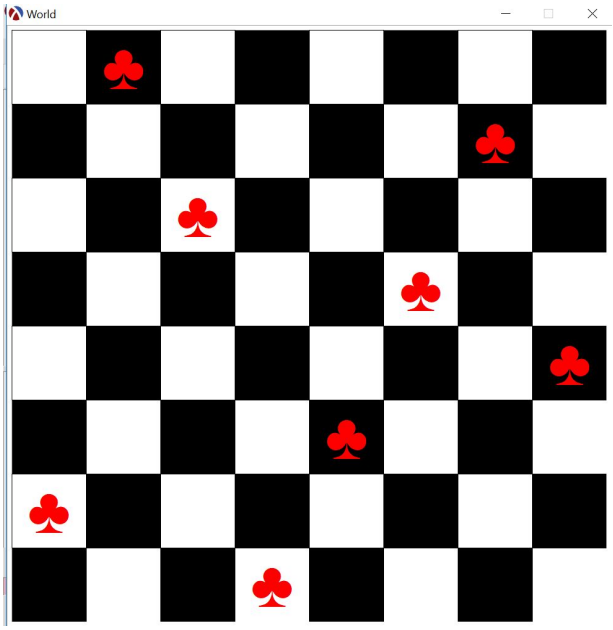
   b. Input is a tree representation of a boolean formula, along with initial assignments to variables

   c. Output is either #f, if the formula is unsolvable or a single solution of the given formula. (We also have written a program that generates all solutions of the boolean formula)

2. **Sudoku Solver:** This code applies the implemented SAT solver to solve the Sudoku. Another optimisation that we have applied here is specific to the problem. On assigning a variable, we have annihilated all the variables that are obsolete as well as as false. This trims down the tree continuously and returns an answer pretty quickly.



3. **N-Queens:** This code also uses the implemented SAT solver and returns positions on a NxN chessboard where queens can be placed, such that they cannot attack one another.



4. **Tournament Scheduler:** The process of making the league fixtures of a tournament is also a boolean problem, and hence can be solved using a SAT Solver. There are many constraints that are needed

to be followed while deciding fixtures, eg, A team should not have matches for consecutive days. Input for this problem is the number of teams playing and we also have kept optional inputs, like any additional constraints on the schedule or the number of days.

5. **Use of Abstractions and Other Syntactic Sugar**

a. Heavy use of Macros like for, while, list-comprehension, in generating the Boolean formula.
b. Use of structures for a general tree representation to solve the SAT formula.
c. Use of a higher order function to make specialised nodes for a list a variables. (only one variable should be uniquely true)
d. Use of data structures like 2d-vectors to represent problems in Sudoku/ NQueens and use of hash-tables in the SAT Solver
e. Graphics to represent the solution of Sudoku and NQueens

Problems Faced

The SAT solver is a famous NP-Complete problem, and as a result there is no polynomial time solution to determine the satisfiability of any CNF equation. However we can use heuristics to improve the speed in some cases. Heuristic methods are not proven and don't always guarantee a fast solution. The problem in consideration have a very large amount of variables. The sudoku has 729 variables while scheduler 1568 variables for 8 teams(near exponential order clauses) in the CNF equation. Simple backtracking will be highly inefficient

We have implemented random variable selection to improve the efficiency. Apart from that, few optimisations such as:

1. Pure Literal assignment: Here unit literals are literals with only either positive or negative variables in all clauses. Such unit literals are assigned such that positive literals are assigned true and negative literals are assigned false to make the overall value true
2. Unit literal Propagation: If a clause has only 1 variable then it will be assigned true. Similarly if all but 1 variables in the clauses are assigned true

References

1. https://www.cs.jhu.edu/~jason/tutorials/convert-to-CNF.html
2. https://www.satisfiability.org/SAT04/programme/74.pdf