Assignment 1- Sudoku pair: Solving and Generation

Dishay Mehta (200341)
Aryan Vora (200204)

February 2, 2022

Abstract

We will be describing the implementation, assumptions, and limitations of both parts of the Assignment.

Part I

Part I of the Assignment asked us to write the code for a k^2-sudoku pair solver by encoding the problem to propositional logic and solving it via a SAT solver.

Implementation:

We have used Glucose3 SAT solver in our code.
The logic of our code is explained as follows

1.Encoding The Given Sudoku

A Sudoku puzzle can easily be represented as a SAT problem, albeit one requiring a significant number of propositional variables. Nevertheless, encoding Sudoku puzzles into CNF requires (k*k)*(k*k)*(k*k) = k^6 propositional variables. For each entry in the (k*k)x(k*k) grid S, we associate k*k variables. Let us use the notation S_{xijy} to refer to variables. Variable S_{xijy} is assigned true if and only if the entry in sudoku x$\in\{0,1\}$ (as we have 2 sudokus), in row i and column j is assigned number y. Hence, $S_{0483} = 1$ means that S[0,4,8] = 3. For a simplicity we have done a one to one mapping of variables as follows

$$S_{xijy} = y + j * (k^2) + i * (k^4) + x * (k^6)$$

. This is basically the conversion of the index in base k^2 to base 10 representation.

2.Adding Clauses And Constraints For Filling Up Sudoku

Initially, we will assign 1 to the literals which already occupy the sudoku. We do this by adding a clause to our CNF to only those variables which are non empty. The constraints to be added to our SAT encoding refer to each entry, each row, each column and each 3x3 sub-grid.
For exactly 1 number in each entry: We have added two clauses each as we want conjunction of atleast 1 number in each entry and atmost 1 number in each entry.
For each number appears exactly once in each row: We have added another two clauses each as we want conjunction of each number to appear atleast once and atmost once in each row .
For each number appears exactly once in each column: We have added another two clauses each as we want conjunction of each number to appear atleast once and atmost once in each column.
For each number appears exactly once in each kxk sub-grid: First we have chose the sub-grid index from the range ([1,k],[1,k]) then we are choosing the cell index in that sub-grid from the range ([1,k],[1,k]). Similar to the previous parts we add two clauses each a number must appear atleast once and atmost once in each sub-grid.

3.Applying The Condition Given In The Question

This is the final step of our approach wherein we have added a clause which checks that $S_{0ijy} \neq S_{1ijy}$.

Limitation:

Takes long time for k more than 6.

Assumption:

An unsolvable sudoku as the input will give None as the answer.
The user goes through the README.md file and understands how to correctly run the files and view the outputs.

Part II

Part II of the Assignment asked us to write the code to generate an unsolved k^2-sudoku pair by encoding the problem to propositional logic and solving it via a SAT solver.

Implementation:

We have used Glucose3 SAT solver in our code.
The logic of our code is explained as follows

1.Generating A Completely Sudoku

We initialize a sudoku with zeros and pass it to the solver implemented in Part I. As SAT Solvers are deterministic the solved sudoku will always be the same given a particular k. We will introduce some elements of randomness into this later.

2.Generating Holes In The Sudoku

Once we have the completely filled sudoku, we create a list of the cell numbers from 0 to $2k^4 - 1$. We randomly shuffle this list and choose the first element of the list. Then, we set that element to be zero in a copy of the sudoku and solve our model and using the enum_models() we count the number of models that are possible with it. If it is unique, we remove that element from the original sudoku as well as the list. We repeat this until the list is empty. If the count of satisfiable models is more than one, we do not remove the number from the original sudoku, but we do remove it from the list.
Hence, we have the maximum possible zeros in the sudoku for it to have a unique solution.

3.Introducing Randomness

Note that due the random shuffle applied on the list, we have introduced a degree of randomness to the sudoku.
Lastly, to introduce more randomness, we generate a permutation of numbers, which we will call shuffled, from 1 to k^2. Then, we exchange the numbers i in the sudoku with shuffled[i].
Now we have the maximal unique sudoku as demanded by the question.

Limitation:

It is very time consuming especially for numbers greater than and equal to 4.

Assumption:

The user goes through the README.md file and understands how to correctly run the files and view the outputs.