CS202

Mathematics in Computer Science -II

Indian Institute of Technology Kanpur

Assignment 1 Report

2nd February,2022

Harshit Raj (200433)

Shubhan R (200971)

From now, we use the word *grid* to refer to both the sudokus together.

Question 1:

We now show our algorithm to generate a partially filled sudoku. Each cell of the sudoku is determined by **3 parameters**:-

- Row index, Column Index, Number stored in it (which is 0 if cell is empty). Which means there are k^2 x k^2 x k^2 propositions. A proposition is true if number stored in the sudoku cell associated with it matches the number used to generate its proposition index, otherwise false.
- We then used a **hash functio**n to map each cell of each sudoku to a unique number, which will also act as the **proposition indices** for each of our propositions.
- We also wrote a **reverse hash function** to get Row index, Column Index, Number back from the proposition index.

We use **Cardinality Encoding** to generate the CNF. We did this because when we added each and every clause as it is to generate the CNF, the runtime was extremely long for n=6 itself. We instantiated a **CNFPlus** (another type of **CNF**) object, and **for each of the 2 input sudokus, we:-**

- Iterated to every cell and put another inner for loop over the numbers from 1 to k^2 and added the CNF which allowed only one of those numbers to be in the cell. We put the bound for **CardEnc.equals()** as 1 here.
- For each row and then number from 1 to k^2, we iterated along the row and added the CNF which allowed every number to come up exactly once in the row. We put the bound for **CardEnc.equals()** as 1 here.
- For each column and then number from 1 to k^2, we iterated along the column and added the CNF which allowed every number to come up exactly once in the column. We put the bound for **CardEnc.equals()** as 1 here.
- Then iterated to each Sub block of the sudoku, then to each number from 1 to k^2, then over all the cells of the Sub block, and added CNF to ensure every number appears exactly once in the Sub block. We put the bound for **CardEnc.equal()** as 1 here.
- We then add the condition that corresponding cells of each sudoku do not match. We put the bound for **CardEnc.atmost()** as 1 here, as both cells can not have the same number at the same time and therefore, the sum of values of the corresponding propositions cannot be 2.

Then, we added **assumptions for the solver in the code** to make the values of propositions of those cells filled in input sudoku true.

We ran it through a satisfiability solver, used reverse hash function to get Row index, Column Index, Number and printed the result and saved it as csv file if it existed. Otherwise, we printed "This set of sudokus cannot form a sudoku pair"

Question 2:

The word *grid* to refer to both the sudokus together.

Random Partially filled *grid* generator:

The following algorithm is used. We first choose a random number (say **p**, where **p** is less than k^2) of randomly selected integers from 1 and k^2 exactly once. We then fill them in **p** random cells of both the sudokus in the *grid*. As the number of integers filled (**p**) is less than k^2 in the *grid* and as each integer is filled only once, we ensure that there will be no violations of sudoku rules while generating both the partially filled sudokus. This also ensures that the 2 sudokus of the grid form a *sudoku pair*.

We use the random partial sudoku generator described above to generate a partially filled *grid*. We the solve it and get a filled *grid* using the code from question 1.

We use the same functions from Question 1 to validate steps in Question 2. In the required steps, we input both the sudokus from Q2 to the algorithm in Q1.

Once we have a randomly generated completed *grid* as input, we empty one random box, and use this *grid* as the inputs for the code from Question 1 with one extra clause that **at least one element of new grid must be different from original grid**. Then we check if the 2 sudokus in the *grid* forms a *sudoku pair*. We then randomly empty **one more** filled cell from the *grid* and validate if it forms a *sudoku pai*r. We keep going on like this till the *grid* does not form a *sudoku pair*. We then **place the last removed element permanently** back into its old place to make the *grid* be a *sudoku pair* again.

The cell we just placed back is fixed permanently. We again empty another randoml filled cell from the grid, and check if it forms a *sudoku pair*. We keep going like till it doesn't form a *sudoku pair*. **Then we place the last removed element into its old place permanently**. Till all the non-empty cells are **permanently fixed**, we keep iterating in a similar way. Then we output the *grid*, which contains 2 sudokus which forms a *sudoku pair* and has the maximal number holes to a csv file.