Q.1 Solver

The Minisat22 Solver is used to implement the code. The code checks every possible value from 1 to k^2 in the empty positions and outputs the suitable solution.

A csv file contains $2k^2$ rows and k^2 columns. The first k^2 rows correspond to the sudoku-1 and next k^2 rows correspond to sudoku-2.
The code first takes the csv file as the input, traverses through it and stores it in a 2-D list named “rows”. (To take it as input, use the appropriate name of the file).
Then it calls the formula_generator function and outputs a set of clauses necessary to create the required sudoku.
In the formula_generator function, we apply different conditions to the sudoku in the form of clauses.
We check that each cell of both sudokus has an element between 1 to k^2 (both inclusive) .
Then in both the sudokus, we check that now two elements in a row are the same and no two elements in a column are the same.
Then we check every small k x k grid of the two sudokus that it contains all the elements from 1 to k^2 .
Finally, we check all the corresponding cells (i,j) of two sudokus and force the condition that $v1(i,j) \neq v2(i,j)$ i.e. value in (i,j) in sudoku-1 is not the same as value in (i,j) in sudoku-2.
After this we return the Solver with all the classes from the formula_generator function.
Then we solve the sudoku on the basis of our constraints and some assumptions(values which are already filled in the sudoku).

If the solver is successful in solving the problem, we create a new 2-D list and put the solved sudoku in the list after decoding all the results.
Then print both the sudokus.
Else, Print “None” .

Q.2 Generator

Here, we enter an input parameter k in the terminal. Then, we generate clauses to get 2-solved sudokus using the formula_generator of Q.1.
Then, we take all the +ve elements of the model generated by the solver and then fill the sudoku by decoding the elements in the model.
Then we create two lists of lengths k^2 and $2k^2$ and randomize them.

After that we run through sudoku1 and sudoku via the randomized lists. In these iterations, we call the cvr(can be removed) function. It ensures whether a value in a particular cell can be removed or not. If on its removal, we can fill the location by another number, then we must not remove it because then the solution won't remain unique.
Similarly we iterate for each of the cells in the two sudokus and generate a sudoku with maximal holes, but having a unique solution.

Note: Some test-cases may take more than the usual time to run and give the output.