Sudoku pair: solving

Implementation:
Input: Value of k, and two partially filled sudoku of size k.
Output: Filled sudoku pair (if it exists) or else 'None'
Constraints:

- Every row of both sudoku must have elements from 1 to k^2 each exactly one time.
- Every column of both sudoku must have elements from 1 to k^2 each exactly one time.
- There are k^2 boxes of k x k and each one of them must have elements from 1 to k^2 each exactly one time.
- Cells(i,j) must have different elements in both sudoku.

Firstly for SAT solver we have to assign numbers as variable so we use the following encoding for the proposition c_{ijy} (cell (i,j) of 1st sudoku has element y)

$$c_{ijy} = i*(k^4)+j*(k^2)+y$$

Similarly, d_{ijy} (cell (i,j) of 2nd sudoku has element y)

$$d_{ijy} = k^6 + i*(k^4)+j*(k^2)+y$$

So by this method for every number in each cell we can have a unique number representing whether it is present or not.
From now on we will represent a cell(i,j) with value y as c_{ijy} or d_{ijy}
Now we write each condition in logical form and for that we have to use the following encoding:

- Each cell must have at least 1 element.
- Every cell in a row must have at most 1 element.
- Every cell in a column must have at most 1 element.
- Every number appears at most once in each subgrid of size k x k
- No two elements in the sudoku pair at the same cell have the same value.

The first two conditions will imply that our solution will have exactly 1 element in each cell. So we need not consider that condition separately.

Now, we give the encoding for the above conditions:

1. Each cell must have at least 1 element:
   $\forall_{i,j}\, c_{ij1} V c_{ij2} V c_{ij3} \ldots\ldots\ldots V c_{ijk^2}$
   $\forall_{i,j}\, d_{ij1} V d_{ij2} V d_{ij3} \ldots\ldots\ldots V d_{ijk^2}$
2. Each row must have each element at most 1 time:
   For each element y in row i:

$\forall_{j1, j2\ (j1 \neq j2)}\ \sim(c_{ij_1y} \wedge c_{ij_1y})$

$\forall_{j1, j2\ (j1 \neq j2)}\ \sim(d_{ij_1y} \wedge d_{ij_1y})$

3. Each col must have each element at most 1 time:
   For each element y in col j:
   $\forall_{i1, i2\ (i1 \neq i2)}\ \sim(c_{i_1jy} \wedge c_{i_2jy})$
   $\forall_{i1, i2\ (i1 \neq i2)}\ \sim(d_{i_1jy} \wedge d_{i_2jy})$

4. Every element appears at most once in each subgrid of size k x k

$$\bigwedge_{z=1}^{k^2} \bigwedge_{i=0}^{k-1} \bigwedge_{j=0}^{k-1} \bigwedge_{x=1}^{k} \bigwedge_{y=1}^{k} \bigwedge_{a=y+1}^{k} (\sim c_{(ki+x)(kj+y)z} \vee \sim c_{(ki+x)(kj+a)z})$$

$$\bigwedge_{z=1}^{k^2} \bigwedge_{i=0}^{k-1} \bigwedge_{j=0}^{k-1} \bigwedge_{x=1}^{k} \bigwedge_{y=1}^{k} \bigwedge_{a=x+1}^{k} \bigwedge_{b=1}^{k} (\sim c_{(ki+x)(kj+y)z} \vee \sim c_{(ki+a)(kj+b)z})$$

$$\bigwedge_{z=1}^{k^2} \bigwedge_{i=0}^{k-1} \bigwedge_{j=0}^{k-1} \bigwedge_{x=1}^{k} \bigwedge_{y=1}^{k} \bigwedge_{a=y+1}^{k} (\sim d_{(ki+x)(kj+y)z} \vee \sim d_{(ki+x)(kj+a)z})$$

$$\bigwedge_{z=1}^{k^2} \bigwedge_{i=0}^{k-1} \bigwedge_{j=0}^{k-1} \bigwedge_{x=1}^{k} \bigwedge_{y=1}^{k} \bigwedge_{a=x+1}^{k} \bigwedge_{b=1}^{k} (\sim d_{(ki+x)(kj+y)z} \vee \sim d_{(ki+a)(kj+b)z})$$

5. For corresponding position elements in the two sudokus
   $\forall_{i,j,y}\ \sim(c_{ijy} \wedge d_{ijy})$
   Where c is for sudoku 1 and d for sudoku2
6. Given cell:
   Some cell are fixed say i,j has element y in sudoku 1, so the encoding is c_{ijy}
   Same for sudoku2: d_{ijy}

Assumptions-We have input with no empty line in between the two sudokus in the CSV file and empty cells are represented as 0.

Limitation-

This works only for k <= 5 (we can improve performance using cardinality enoder, a PySAT function in the pysat.card library to make it work upto 7 but we prefer to use encoding manually instead of cardinality encoder over here). After k=5, we were unable to test on our systems since they ran out of RAM.

Sudoku pair: generation

Input- We have been given value of k
Output- We have to generate the partially filled sudoku pair which has a unique solution and is maximal.
Constraint-The sudoku-pair must have a unique solution with the solution having conditions same as above and also it should have maximal numbers of holes in it.
Approach-

1. Firstly we used the same encoding as before and then we tried to remove one element at a time and then created a new solver to check whether it has a unique solution or not but it is taking too much time ever for k=3.
2. Then we got to know about cardinality encoders so we tried to implement it and to get the extended encoding(explained below) and then we had somewhat reduced the time but still we tried to reduce it more.
3. So instead of taking a new solver each time, we used a different approach. We give the argument that lets say we have the list of elements that we have removed till now and we are on the step to remove the next element. We give our solver as assumptions, the list of elements that are left in the sudoku pair except the current element and the negation of the current element. So, we are saying that whether there exists a solution where there is some other element in place of the current element, if yes then the removal of the current element will cause multiple solutions which we don't want so we skip it.
4. Now we are able to solve for k=3,4 but still we wanted more optimisation, so we changed the approach and instead of removing elements from a filled sudoku we generated a random sudoku and **then start randomly filling the element in empty sudoku** until we get to the condition **of unique solution** and then we try **removing the filled elements** (if they give a unique solution even on removal) so that we can get maximal holes.
5. We found that the last approach is much faster than the previous ones so we adopted it. And the reason that we thought is that the maximal sudoku has more holes than filled cells so it is better to fill the sudoku instead of emptying it.

Extended encoding:

1. Each cell must have only 1 element.
2. Each cell in row must have all elements from 1 to k^2 one time only.
3. Each cell in the column must have all elements from 1 to k^2 one time only.

4. Each cell in a subgrid of size kxk must have all elements from 1 to k^2 one time only.
5. The pair must have different elements in the same cell.

Implementations-

1. First of all we fix the corner values of the sudokus using a random seed generator.
2. Then we use the 1st part (sudoku solver) to solve the sudoku for us without any filled boxes.
3. We make a list of all the cell numbers from 1 to $2k^4$. Then we randomly shuffle the list.
4. Then we iterate through the list and start putting the element in the empty sudoku and check whether we have a unique solution of the given sudoku or multiple solutions.
5. We keep on repeating step 4 until we get a sudoku with a unique solution but now it is not guaranteed that it also has a maximal number of holes.
6. So now we keep on removing elements whose removal still guarantees unique solution.
7. So we get the answer and we print it into a csv file and also onto the terminal

Assumptions-We have printed in csv with no empty line in between the two sudokus in the CSV file and empty cells are represented as 0.

Limitation-

This works only for k <= 4. After that, we were unable to test on our systems since they ran out of RAM.