Assignment 1

CS202A, Group 18

Uttam Kumar | 201071 | uttamk20@iitk.ac.in
Kavya Jalan | 200503 | kavyajalan20@iitk.ac.in

Question 1

Given a sudoku puzzle pair S1, S2 (both of dimension k) as input, your job is to write a program to fill the empty cells of both sudokus such that it satisfies the following constraints,
1.Individual sudoku properties should hold.
2.For each empty cell S1[i, j] ≠ S2[i, j], where i is row and j is column.

Input: Parameter k, single CSV file containing two sudokus. The first k*k rows are for the first sudoku and the rest are for the second sudoku. Each row has k*k cells. Each cell contains a number from 1 to k*k. Cell with 0 specifies an empty cell.

Output: If the sudoku puzzle pair doesn't have any solution, you should return None otherwise return the filled sudoku pair.

Answer: IMPLEMENTATION:

1. We need to reduce the sudoku problem to a set of clauses in conjunctive normal form which can be fed to the sat solver.So ,we need to put a valid set of constraints to our sat solver.

2. **Determining number of variables**
   Let the value of **T = K*K**.
   There will be be total T*T elements in one sudoku and every element can take values between [1,T]. Therefore, total number of variables required will be 2*T*T*T. We assign first T*T*T variables to the first sudoku and next T*T*T variables for the second sudoku.
   eg. If **sudoku1(i,j) = d** (d ∊ [1,T]) then variable associated to it will be **T*T*(i - 1) + T* (j - 1) + d**
   for **sudoku2(i,j) = d** it will be **T*T*T + T*T * (i - 1) + T* (j - 1) + d**.

If **sudoku1(i,j) ≠ d** must hold then negation(-) of that variable should be added to constraints.
We can get the variables easily by this code:-

```
def v1(i, j, d):                                                  for first sudoku
.          return T*T*(i - 1) + T* (j - 1) + d
def v2(i, j, d):                                                  for second sudoku
.         return T*T*T+T*T * (i - 1) + T* (j - 1) + d
```

3. Now we need to determine all the constraints in form of clauses that is enough to ensure that out solution holds.

4. **CONSTRAINTS**

   (a) **Each cell must contain at least one integer ∈ [1,T]**
   This can be done by adding clause [v1(i,j,d) for all d∈ [1,T]].Actually adding this clause will ensure that cell is assigned to at least one value among[1,T]. We need to do it for all values of i and j in range[1,T] for both sudokus.

   (b) **Each cell must not contain two different values at once**
   We ensure this by adding [-v1(i, j, d), -v1(i, j, d1) for d1 ∈ [d+1,T]].This clause will ensure that the same cell doesn't contain both d and d1 value. This should hold for all cells and both sudokus .

   (c) **Corresponding cells of the sudokus must not contain the same value**
   This is ensured by adding the clause [-v1(i,j,d),-v2(i,j,d)] to our constraints.

   (d) **Ensure rows and columns have distinct values**
   This is done as follows:-
   Choose every pair of cells in a row and ensure they contain different values. Similarly for every columns.

   (e) **All sub grids must have distinct values**
   To do this choose every pair of cells in a sub grid and ensure these cells doesn't contain the same values.

   (f) **Values which are already filled must be added to constraints**
   We can iterate through all cells and if it contains non zero value then the corresponding variable will be added to the constraints.

5. These set of constraints will be fed to the sat solver. It will return us a model to our constraints or tell us if it is not possible to achieve the goal. Then from the model obtained we can encode it get our solution by transforming every valid variable in solution to corresponding sudoku cells and the value it will contain.This operation is done by using readcell(i,j) in our code.

Question 2

In the second part, you have to write a k-sudoku puzzle pair generator. The puzzle pair must be maximal (have the largest number of holes possible) and must have a unique solution.
Input: Parameter k
Output: CSV file containing two sudokus in the format mentioned in Q1.

2A : Approach

Answer: A way to generate a sudoku puzzle is to take a solved sudoku and keep removing numbers from cells such that the puzzle has a single solution. We must also ensure that the puzzle pair generated is maximal.

How to do it?

Answer: Start with a list which contains numbers [0,1,...,2*T*T-1] (note :T=k*k). The numbers in this list represent the cell of the sudoku and hence it's size is equal to the total total number of cells in both sudokus. Numbers from [0,T*T-1] represent the cells of the first sudoku and the numbers form [T*T,2*T*T-1]represent the cells of second sudoku. eg. '0' in list denotes Sudoku1[0,0] and 'T*T' in list will represent Sudoku2[0,0].

Now choose a random integer∈[0,len(list)-1]. Let us say that random integer was X. Then make the value of the cell represented by list[X] equal to '0'(means empty the cell). Now check the number of solutions this sudoku will have. If the number of solutions are greater than 1 then undo the process. Finally, remove the element at index X from the list.So,that the length of list decreases by one in every operation. Keep repeating the process until the list becomes empty.

Does it ensure that the puzzle pair generated is maximal?
Yes, this process ensures that the generate puzzle will be optimal. Because, we went to each and every cell of both sudoku. We removed the value it contained if after its removal there was still a unique solution.

2B: How to get a Solved Sudoku?

Answer: A way to do it is initialize a blank pair Sudoku and solve it using our code and then use it. But a problem arises, Our sat solver will give the same solution every time we run it. So, we will be able to generate different puzzles which will also be maximal but they will have the same solution. This will meet our requirement but puzzles will become boring.

How to overcome that?

Answer: Start with a blank sudoku.Randomly fill some cells of the sudoku such that it remains satisfiable. Now , solve this sudoku to get a solved sudoku. I found that doing this produced fairly different solutions. Now , we can use this to generate our puzzle.

An even better way get the random solutions to the blank sudoku was to keep filling the cells randomly so that it remains satisfiable. We would stop only after there existed only one solution to our sudoku. Now solve this puzzle to get a solved pair sudoku. This process would take time.

I found that initializing only a few cells was enough to generate different solutions. This was also a faster way.

ASSUMPTIONS AND LIMITATIONS

1. The inputs provided to the solver must be in proper format. The number of inputs lines must be 2*k*k and each line should contain k*k numbers.

2.Generator Code might take hours if value of k is large(>7).

3. I tested my pairgenerator code only for k≤5 and assumed it till run correctly for larger values.