Implementation

Defining up Variables

We define variable x({1,2})(row no.)(col no.)(value)

fac=10**(int(math.log10(k*k)+1)) # factor by which row no., col no., value will be separated

Variable name={1,2}*fac**3+i*fac*fac+j*fac+cell_value # (S1,S2) ; (ith row) ; (jth column) ; (value of cell)

Applying Constraints

We have applied constraints for the k-sudoku pair by using Cardinality Constraints (CardEnc) provided by PySAT. Given below is the list of applied constraints.

1) **Value Constraints** - Each cell must have only one value from 1 to k*k for both S1, S2
2) **Horizontal & Vertical Constraints** - each column and row must have exactly one occurrence of each number from 1 to k*k
3) **Block Constraints** - each sub-block must have only one occurrence of every number from 1 to k*k
4) **S1[i,j] != S2[i,j]** - Corresponding cells in S1, S2 must not be equal. Atmost one is possible.
5) **Diagonal Constraints [optional]** - both overall diagonals should have numbers ranging from 1 to k2 exactly once - for a m, varying each cell in a diagonal, summation (xa,i,j,m) = 1

Sudoku Pair Solver

Above constraints are added to the CNF formula which is then appended to Solver. An assumptions list is created using "assume_gen(rows,k)". Then, we get the corresponding model. After that, we use the get_ans() function to get a fully filled sudoku from the model.

Already filled values are added to corresponding variables and passed to Solver as assumptions

Sudoku Pair Generator

An empty Sudoku is taken. Then its diagonal elements are filled with random numbers till at least one solution of sudoku is possible like (left-top in S1 and bottom-right in S2) block-wise using ran_fill(). Then, sudoku is completely filled using the solver of part-1. Then, We start removing any random element from the sudoku-pair if sudoku is not satisfied by any other value in the cell from 1 to k*k. We ran this for every single cell in sudoku.

Files Used

Optional Files:

1) **Performance_check.py:** This prints the time taken to run for the various values of k. And appends the output to Performance_log.txt. This checking ignores diagonal constraints.
2) **Print_sudoku.py**: This provides functions to print the sudoku in a user-friendly format from the list of list
3) **Sudoku_checker.py:** This script checks all .csv files in the test_cases directory for already invalid sudokus.

Core Files:

1) **Sudoku_solver.py:** This includes core functions for solving sudokus like formula generator, assumption generator, and sudoku generator from the model.
2) **Sudoku_generator.py:** This includes core functions for generating sudokus like can_remove() which decides if an element can be removed from the given cell.
3) **Print_sudoku.py:** This includes functions that have the capability to print sudokus from the given list of lists in various formats.

Interface Files:

1) **Main_solver.py:** It interacts with the user to get input CSV files and other parameters and extract solutions from core files.
2) **Main_generator.py:** It interacts with the user to get parameters and print output to the terminal

Assumptions

- All python scripts are in the same folder
- There is a folder named test_cases in the same directory of python scripts
- PySAT is installed on the system and includes pysat.solvers, pysat.card, pysat.formula
- Other modules include math, glob, csv, random

Limitations

- It takes a lot of time for generating a k-sudoku pair puzzle for k >5.
- The command-line utility is yet to be added (in terminal input is used).