Contributors

Kushagra Sharma (200539)
Mandar Wayal (200556)

Part 1: k-Sudoku Pair Solver

Brief Logic:

We create $2k^6$ variables for our encoding (1 to $2k^6$). There are total k^4 cells for each sudoku, and each cell has k^2 possibilities for values (i.e. [1,k^2]).
Let (i, j) represent the cell address in zero based indexing (i.e. i, j take values from [0,$k^2 - 1$]).

Now, if a cell (represented by (i, j)) in *sudoku1* has a value v present in it, the corresponding variable can be found through the formula:
$correspondingVar(C_{ijv}) = i * k^4 + j * k^2 + v$
Similarly, for a cell (represented by (i, j)) in *sudoku2* has a value v present in it,
$correspondingVar(C_{ijv}) = k^6 + i * k^4 + j * k^2 + v$

If the value v isn't present in the cell, it will be represented as $(-\ correspondingVar)$.

Now, all that is left is to create a CNF formula and provide it to the SAT solver.

Let's list out the constraints:

1. Any **row** must contain all **unique values** (from $\{1, 2, \dots, k^2\}$)
   For a fixed row number i,
   - Each value comes at least once in the row *and*
   - Each value comes at most once in the row

   Note: The corresponding encodings can be easily converted into CNF forms, which can be seen in the code.
2. Any **column** must contain all **unique values** (from $\{1, 2, \dots, k^2\}$)
   For a fixed column number j,
   - Each value comes at least once in the column *and*
   - Each value comes at most once in the column
3. Each **box** must contain all **unique values** (from $\{1, 2, \dots, k^2\}$)
   For a fixed box,
   - Each value comes at least once in the box *and*
   - Each value comes at most once in the box

4. Each **cell** must have only **one value**
   For a fixed cell (i.e. fixed (i, j)),
   - C_{ijv} should only be true (positive in this case) for exactly one value of v.
5. The **corresponding cells** in the two sudokus must have **different values**.

All these constraints are added in the form of clauses as shown in the code and fed to the SAT solver.
If a solution exists (i.e. the encoding is satisfiable), the SAT solver gives a model satisfying the encoding (which we print).
If not, we print "None".

Assumptions:

- The first k^2 rows are for the first sudoku and the rest are for the second sudoku. Each row has k^2 cells. Each cell contains a number from 1 to k^2. Cell with 0 specifies an empty cell.

- k has to be given as input from the terminal and not through the CSV file. The program prompts and asks for k.

Limitations:

- As the value of k increases, the time taken increases drastically. This is because there are $2k^6$ variables and the clauses are of the order k^8 .

Part 2: k-Sudoku Pair Generator

Brief Logic:

How did we generate randomness?

We took two empty sudokus (all cells filled with zeros). The plan was to fill it with some numbers and then make the program for Part 1 give two solved sudokus.
For *sudoku1*, we filled the first cell of each diagonal box with a random value chosen from 1 to k^2.
For *sudoku2*, we filled the last cell of the first row of a box with a random value chosen from 1 to k^2.
This ensures the property of a sudoku as we don't fill the same number in the same row, column or box. As we fill different diagonals in the two sudokus, same number doesn't appear in the corresponding cells.

It can be proven that there exist $at\ least\ k^{(2*k)}$ different such sudokus at this point. This is because there are k boxes in a diagonal, (so k cells are filled) and each cell has k^2 different values (assigned randomly). These partially filled Sudokus are then completed using the code in Part I of the assignment, which would give a single random solution out of the many possible solutions.

Further in the code, we also remove the values from the cells randomly. So, there are certainly many more such possible sudokus. This would be more clear later.

Explanation

- After generating two sudokus with some values filled, we feed it to the first part of the assignment. It gives out two solved sudokus.
- The plan is to remove values from random cells and check for solutions.
- We created a list of number in the range $(1, 2k^4)$. Each number represents a cell in the sudokus.

  If the number (say n) is less than or equal to k^4, the cell is from *sudoku1* and it's coordinates are given by (x,y) where

  $x = floor\ (n/k^2)$ $\quad\quad\quad\quad$ $y = n\%k^2$

  If the number (say n) is greater than k^4, the cell is from *sudoku2* and it's coordinates are given by (x,y) where

  $x = floor\ ((n - k^4)/k^2)$ $\quad\quad\quad\quad$ $y = n\%k^2$
- The list is now shuffled randomly. Now, when we remove cells corresponding to the values that appear in the list, it creates randomness.

How is it a maximal sudoku pair?

- We iterate through the list, replace the value in the corresponding cell with 0 and check if multiple solutions exists.
- If multiple solutions exists, we place the value back and move ahead to remove the value from the next cell in the list.
- If there is a unique solution after the removal, we move ahead without replacing the value (i.e. a hole is created).
- After we reach the end of the list, we have tried removing all the cells. The only values left are the ones on whose removal, the sudoku pair started having multiple solutions. The pair would still give multiple solutions if any of the filled cells are removed.

  Thus, it is maximal.

How did we check if multiple solutions exist?

- If the encoding is satisfiable, we store the model, negate it and append this clause to our existing encoding.
- Now, if it is satisfiable, we know that the current pair has multiple solutions.

Limitation:

- As we check for model each time after creating a hole, it takes a lot of time to generate the required pair for large values of k.