CS202A: Mathematics for Computer Science -II

Even Semester, 2021-22

Assignment – I

180771 Soham Ghosal *190189 Aryash Pateriya*

***Languages/tools used**: Python3, MiniSat*

Overview: Sudoku

A normal sudoku is a mathematical puzzle in a tabular form which consists of 9 different rows and 9 different columns containing 81 cells. The sudoku table is further divided into 9 3x3 squares containing 3 rows and 3 columns each, say sub-grid.

Some of the cells are pre-filled with integers ranging from 1 to 9. Solving a sudoku puzzle refers to filling an integer ranging from 1 to 9 in every non-filled cell such that it satisfies the constraints that no two cells across a row, a column or a sub-grid can have the same integer. If such a condition is met, we say the sudoku is solvable and the proposed combination is a solution for it.

k-sudoku

A k-sudoku is a variation of the original sudoku puzzle such that the puzzle consists of k^2 rows, k^2 columns, and k^2 sub-grids of the dimensions k^2 x k^2. The cells of a k-sudoku can contain an integer ranging from 1 to k^2.

The other rules, when compared with a normal sudoku, remain the same. Some cells of a k-sudoku can be prefilled with integers 1 to k^2, and the goal is to fill every non-filled cell of a sudoku with some integer in 1 to k^2 such that no two cells of a row, or a column, or a sub-grid can contain a similar value. The solution to a k-sudoku is a valid combination filled in the cells such that it meets all the constraints above mentioned.

k-sudoku pair

A k-sudoku pair is defined as a puzzle consisting of two different k-sudokus, which is considered satisfiable if all the unfilled cells of both the k-sudokus can be filled with the integers ranging from 1 to k^2 such that each of the rules of a k-sudoku are individually satisfied by the proposed combination of integers of a cell along with an additional constraint that for every cell of the first k-sudoku having a particular index (row number, column number), should not contain the same integer in the same index of the second k-sudoku.

If the given condition can be satisfied, we say the k-sudoku pair is satisfiable, otherwise it is unsatisfiable.

Task 1

Write a program using a SAT-solver in python to fill the empty cells of a given k-sudoku pair which should print a valid solution for the given input or otherwise informs if the given k-sudoku pair is unsatisfiable

Given: A CSV file containing the information of a k-sudoku pair, the parameter 'k'

Implementation:

We start by defining a total of $2*(k^6)$ premises which contains the information if a particular cell among all the k^4 cells of the first and the k^4 cells of the second sudoku contains any integer ranging from 1 to k^2.

In our code, we represent the cell at the i^{th} row and the j^{th} column of the first k-sudoku with the integer $(k^2 * i) + j+1+(k^4) * (q-1)$ as the variable if the cell contains the integer 'q' ($1<= q <=k^2$). The cell at the i^{th} row and the j^{th} column of the second k-sudoku is represented with the integer $(k^2 * (i-k^2)) + j +1+(k^4) * (q-1)+k^6$ as the variable if it contains the integer 'q' ($1<= q <=k^2$).

Take an example, where we have a 3-sudoku pair. Let's say we have the integer 3 at the 2^{nd} column of the 1^{st} row of the 1^{st} 3-sudoku. Then the variable $3^2 *1 + 2 + 1 + 3^4 * (3-1) = 174$ stores this information. Similarly, if we have 3 at the 2^{nd} column of the 1^{st} row of the 2^{nd} 3-sudoku, the information is represented using the variable $(3^2 * (1-3^2)) + 2 +1+(3^4) * (3-1)+3^6 = 822$.

We generate an encoding by writing the code for ensuring whether each row, each column, and each sub-grid of the individual k-sudoku contains all the integers in the range 1 to k^2 exactly once and none of the two cells having the same index of row and column, one for the first and other for the second k-sudoku contains the same integer.

We then add the information of the prefilled cells of the given k-sudoku pair in our encoding as well. These encodings are written in the CNF form. We then take the help of a SAT Solver, minisat, which checks for the satisfiability of the provided encoding. If the given k-sudoku pair is satisfiable, we get a valid solution of the k-sudoku pair. If the pair is unsatisfiable, then none is returned.

Running the program:

Requirements: *Python3 setup with minisat installed*

There are initially 5 files provided in the folder. The input.csv file contains the information of the k-sudoku pair where the first k^2 rows contain k^2 cells representing the first k-sudoku and the next k^2 rows represent the rows of the second k-sudoku. Empty cells are represented using the integer value, 0.

To begin with, one should run "make". This runs the 'run.sh' script, which takes care of the rest.
It first calls generate_encoding.py file which takes the parameter 'k' and the input.csv file as input and outputs encoding_CNF.txt file which contains the information of the generated encoding in the CNF form in the format suitable for the minisat to read. The minisat provides the output in the solution.txt file.

Finally, the generate_output.py file helps in converting the minisat output in a user-friendly readable format by providing the output.txt file.

Task 2:

Write a program using a SAT-solver in python to generate a minimal k-sudoku pair

Given: Parameter 'k'

Implementation:

We maintain an "allowed" array, initially containing all variables. We randomly select a variable from this array, and try to put it in the corresponding sudoku. Now, we check the number of solutions of the current sudoku pair.
If there are multiple solutions, then we update the "allowed" array. Now, the allowed array has shortened.
If there is a unique solution, we have got the required answer.

We do this until we get a unique solution for the selected values.

Now, in-order to get the minimal sudoku, we remove every number filled till now, one by one, to check whether the current k-sudoku pair has a unique solution. If there are multiple solutions, then we put the number back, else we remove it permanently.

Note that we check for multiple solutions by providing the negation of the current solution as an extra condition to the solver. If the pair can still be satisfied, then we can have multiple solutions, else we can have only one solution for this case.

Running the program:

Requirements: *Python3 setup with minisat installed*

There are 2 files provided in this folder. To generate a random minimal k-sudoku pair, simply run "make" and fill in the parameter 'k' as the input. The resulting minimal k-sudoku pair is obtained as an output.csv file.

Sample test cases are provided in the "tests" folder.

Time taken to generate sudoku pairs during testing:
k=2:Instantly, k=3:<1 min, k=4: <7 mins

Assumptions:

1. k=1 will not be given as an input to the generator. This is because no such sudoku pair exists.
2. The csv file given as an input to the solver will be of dimensions 2*(k**2) by (k**2).

Limitations:

1. It takes a lot of time to generate sudoku pairs for k=5 or 6, or higher values.
2. The internal "randint" function is not exactly random.(pseudo-random)

References:

1. https://dwheeler.com/essays/minisat-user-guide.html
2. https://www.lri.fr/~conchon/PFA/PROJET/A_SAT-based_Sudoku_solver.pdf