CS202 Assignment

Manas Gupta (200554)

Priya Gole (200727)

1. K-pair sudoku solver:

Sol. Let x denote the variables in sudoku 1 and y denote variables in sudoku 2.

Let the cell in sudoku 1 holding number n in r^{th} row and c^{th} column be denoted as x_{rcn} and let the cell in sudoku 2 holding number n in r^{th} row and c^{th} column be denoted as y_{rcn}.

Now, for sudoku pair to have a solution:

(i) All cells in a row should have distinct numbers.

The formula constraint is:

$\bigwedge_{r=1}^{k^2}\left(\bigwedge_{n=1}^{k^2}\left(\bigvee_{c=1}^{k^2} x_{rcn}\right)\right)$ and $\bigwedge_{r=1}^{k^2}\left(\bigwedge_{n=1}^{k^2}\left(\bigvee_{c=1}^{k^2} y_{rcn}\right)\right)$

(ii) All cells in a column should have distinct numbers.

The formula constraint is:

$\bigwedge_{c=1}^{k^2}\left(\bigwedge_{n=1}^{k^2}\left(\bigvee_{r=1}^{k^2} x_{rcn}\right)\right)$ and $\bigwedge_{c=1}^{k^2}\left(\bigwedge_{n=1}^{k^2}\left(\bigvee_{r=1}^{k^2} y_{rcn}\right)\right)$

(iii) All cells in a box should have distinct numbers.

The formula constraint for top left sub box is:

$\bigwedge_{n=1}^{k^2}\left(\bigvee_{r=1}^{k^2}\left(\bigvee_{c=1}^{k^2} x_{rcn}\right)\right)$ and $\bigwedge_{n=1}^{k^2}\left(\bigvee_{r=1}^{k^2}\left(\bigvee_{c=1}^{k^2} y_{rcn}\right)\right)$

Similarly, we can extrapolate for other sub-boxes.

(iv) Each cell should contain a number.

$\bigwedge_{r=1}^{k^2}\left(\bigwedge_{c=1}^{k^2}\left(\bigvee_{n=1}^{k^2} x_{rcn}\right)\right)$ and $\bigwedge_{r=1}^{k^2}\left(\bigwedge_{c=1}^{k^2}\left(\bigvee_{n=1}^{k^2} y_{rcn}\right)\right)$

(v) Corresponding cells in both sudoku should have different number.

$\bigwedge_{r=1}^{k^2}\left(\bigwedge_{c=1}^{k^2}\left(\bigwedge_{n=1}^{k^2}\left((\neg x_{rcn})\bigvee(\neg y_{rcn})\right)\right)\right)$

We have defined all these rules in a function $sudoku_rules(solver)$, where solver is the initialized pysat solver.

For each x_{rcn} and y_{rcn}, we encoded the statements into unique id using function $encode\_id(val, r, c, k, n, s)$ where val is number of digits in k^2 and s is the sudoku number(i.e. sudoku 1 or sudoku 2).

The id is encoded in the following way:

$$id = s \times 10^{3val} + r \times 10^{2val} + c \times 10^{val} + n$$

For example, say we have $k = 2$, then $val = 1$ (number of digits in $k^2 = 4$ is 1).
Consider the statement: In sudoku 2, in 1^{st} row and 1^{st} column, number 9 is present.
This statement will be encoded as:

$$= 2 \times 10^3 + 1 \times 10^2 + 1 \times 10^1 + 9$$

$$= 2000 + 100 + 10 + 9$$

$$= 2119$$

The most significant place denotes the sudoku number.
The next "val" places denotes the row number and next to those "val" places denotes the column number.
And the last "val" places denotes the number n.
For ease of accessing these unique ids, we stored them in arr1 and arr2 corresponding to sudoku 1 and sudoku 2. (arr1 and arr2 has 3 dimensions, it is generated as a list of list of list).

While taking input, when a cell is occupied, the statement assumes value "True". Then the statement is added as a clause in the solver.

Using these unique ids, we encoded the above formulae and solved it using $Solver()$ function i.e. pre-defined in pysat.

Limitation: As the value of k increases, the runtime increases.

---

2. K-sudoku puzzle pair generator:

Sol. For generating a maximal k-sudoku pair with unique solution, we first take an empty k-sudoku pair and solve it.
Starting with a unique solution, we keep eliminating numbers from random places till we get a sudoku pair with non unique solutions. That is a point where we stop removing the numbers from the pair, giving us a maximal k-sudoku puzzle pair with a unique solution.
The algorithm for implementing the above logic is as follows:
We generate a unique solution by starting with an empty board and solving it with k-pair sudoku solver.
We add the encoded variables of the *get_model* function(statements which are true, the unique ids corresponding to those statements are added) in a new list named solutions.

The variables in this list will be the solution to the sudoku solved. Now, we add the negate of these solution clauses in the solver.

Now we declare a new list *new_enc* which will contain the solution encoding and we use the shuffle method for its randomization. Now, we declare an empty list for taking the final assumptions named *imp*.

Then we declare a list named $flat$ and we flatten array 1 and array 2 into a single list. Now, we assign phases(positive/negative) to random encoding from the flattened list.

We loop through the length of the *new_enc* encoding. On each iteration, we pop an element from the *new_enc* list and store it in enc variable.

We then assume *imp* and *new_enc* to be true clauses and add them in the solver.

If $solver.solve()$ returns true i.e. a SATisfiable solution is found, then we append *enc* into *imp* list else if no alternate solution exist then we drop *enc* and remove the encoding not important for deriving unSATisfiability of our sudoku.

In the end of iterations, only those encoding remain which will give us a unique solution, hence giving us a maximal solution.
