Assignment Report

Nishi Mehta - 200645
Mohd Umam - 200595

Sudoku Pair Solving in SAT Solver.

Implementation

The overall gist of the source code is as follows -

- Scanning the parameter k from standard input.
- Accessing csv file to make sudoku 1 and sudoku 2 as list of list
- CNF of propositional logic that can be provided to sat solver
- If sat solver returns true then we access one of the satisfiable solution from the sat solver else we return “not possible”

The algorithm to make a propositional logic is -

- The boolean variable we introduced is $A_{S,ROW,COL,VAL}$
- A is varies on 4 parameters -
    - Sudoku index i.e. sudoku1 or sudoku2
    - Row index that varies from 0 to 8
    - Column index varies from 0 to 8
    - Values varies from 1 to 9 that each cell can hold
    - So total number of boolean variables used for making propositional logic is 2*9*9*9 = 1458
- We map this 1458 boolean variables to integers starting from 1 to 1458 in such a way -
    - $A_{S,ROW,COL,VAL} = S*k^6 + ROW*k^4 + COL*k^2 + VAL + 1$
    - For example when k==3 some of the variables will be mapped as
    - $A_{0001} = 1$
    - $A_{0002} = 2$
    - $A_{0010} = 10$
    - $A_{0100} = 82$
    - $A_{1000} = 730$
- After mapping the variables, we add clauses to our logic so that -
    - Each cell has exactly 1 value
        - LOGIC - each cell has at least 1 value AND each cell has at most 1 value
    - Each box has all the values exactly once

        - LOGIC - each box has each value at least once AND each value is at most once in a box
    - Each column has all the values exactly once
        - LOGIC - each column has each value at least once AND each value is at most once in a row
    - Each row has all the values exactly once
        - LOGIC - each row has each value at least once AND each value is at most once in a row
    - Both sudoku have different value at same position index
        - Intersection of all the values at same position is false for every position index
    - Boolean variables for those values which are already present at their respective positions has to be true
- Making a propositional logic and feeding it to the sat solver should return true or false whether the logic is satisfiable or not.
- If satisfiable then we access 1 possible satisfiable solution and fill the entries that are 0 in the sudoku list

Assumptions

- We assume that the given unsolved sudoku pair follows the condition that values at the same position are either not filled or unique if they are pre-filled. If this doesn't satisfy there does not exist any sense in solving the propositional logic, no matter what the answer will be not possible.
- We assume that when test cases are checked, the grader will fill the parameter according to the sudoku pair present in the csv file. It could get complex to read a csv file when the first entry is of a single column while all the other entries are extended upto k^2 columns.

Limitations

- First and foremost limitation is that the run time increases as we try test cases of slightly larger k like 5,6. This can be justified by the fact that the number of variables that depend on k^6 and each variable is used multiple times.
- Due to some mathematical inconsideration, if the entry resembling the value k^2 is not filled that is to be evaluated, the code returns 0 instead of k^2, this is tackled by checking if the value in final sudoku 0 and if so replacing it by k^2. This error could not be handled in the best possible way because of time limitations.

Sudoku Pair Generation in SAT Solver.

Implementation

The overview and algorithm to generate sudoku pair is as follows -

For generating a sudoku pair

- First step is to find a unique sudoku pair. 2 ways to do this are -
    - Either find a pre-existing sudoku pair that holds the condition of being unique.
    - Else find a solved sudoku and one to one map its values to itself *($v \rightarrow v$; $v \in z,[1,k^2]$)* And replace the value according to the mapping so that we get second sudoku which is unique.
- Now start removing the values from random positions, first from the sudoku1, then from the sudoku2 and then back to sudoku1 and so on.
    - These random integers can be generated using python library
    - If the randomly generated pair already has a 0, which means that value from this position has already been removed, we discard this position and regenerate a random position index.
- After each removal of value (basically changing its value to 0), try to find if the sudoku pair satisfies only 1 solution. If so, continue, else stop.
    - Checking if there exist only 1 solution can be done by making minor changes in the code of the previous part. Add the clauses of negation of the variables that are removed.
    - If the sat solver returns false, means there exists only a single solution which was the original sudoku pair, but if it returns true then a new solution apart from already known sudoku also exists.
- This means that we go on removing elements randomly one by one from each sudoku until we remove an element which produces a non unique solution for our sudoku pair. The last removed element gave a non unique sudoku pair so that element has to be again placed in the sudoku instead of 0 in that position.

Thus, finding a sudoku pair with maximal elements removed such that the solution is unique.

Assumptions

- The compiler should not have time limits on running the code, else the code will terminate due to TLE.

Limitations

- If we use a limited data set of solved sudoku, for example 1 for each k, the generated sudoku pair may be different while we run the code repetitively, but its solution will be the same each time.

- Due to limitations of the data set, the code can only return a sudoku pair for k being an integer ranging from 2 to 6.