Project - Solving Sudokus

Elie Azoury

Nov 2023

Implementation

Generic Constraints

1. Cell Constraints: Each cell must contain at least one number.
For cell (i, j), the constraint is:

$$\bigvee_{k=1}^{N} x_{i,j,k}$$

2. Row and Column Constraints: Each number k must appear at most once in each row and column.

Mathematically, the constraint can be expressed as:

$$\forall x, y, k \in \{1, 2, ..., N+1\}$$
$$\forall z \in \{y+1, ..., N+1\}$$

$$\text{Then } \neg X(x, y, k) \vee \neg X(x, z, k) \text{ And } \neg X(y, x, k) \vee \neg X(z, x, k)$$

3. Box Constraints: Each number k must appear at most once in each $n \times n$ box. Mathematically, the constraint can be expressed as:

$$\forall k \in \{1, 2, ..., N+1\}$$
$$\forall x, y \in \{1, ..., n\}$$
$$\forall u_1, v_1 \in \{1, 2, ..., n+1\}$$
$$\forall u_2 \in \{u1+1, ..., n\}$$
$$\forall v_2 \in \{\text{v1 + 1 if u1 == u2 else } 1, ..., n\}$$

$$\text{Then } \neg X(n \cdot x + u_1, n \cdot y + v_1, k) \vee \neg X(n \cdot x + u_2, n \cdot y + v_2, k)$$

Specific Constraints

The sudoku start with initial clues, these should also be added as contraintns:

Mathematically, the constraint can be expressed as:

$$\forall x, y, k \in \{1, 2, ..., N+1\}$$
$$\text{Then } \bigwedge_{\substack{(i,j,k) \\ \text{in Clues}}} x_{i,j,k}$$

Check if the solution is unique: Other Constraints

To determine if the solution is unique, we first find an initial solution. If found, we should negate it and add it as a constraint. The SAT solver will then try to find another solution.

$$\bigvee_{\substack{(i,j,k) \\ \text{in solved puzzle}}} \neg x_{i,j,k}$$

N.B.: To handle the case where i, j, or $k > 9$, we append '0' depending on the number of digits in the length of the sudoku to make the encoding of constraints consistent in the SAT solver.

The program was able to solve Sudoku puzzles of sizes 9x9, 16x16, and 25x25 very quickly (0.8s for the 25x25 sudoku) and find alternative solutions.

Generating a sudoku

The process involves a combination of all previously mentioned constraints. Initially, we generate a fully solved Sudoku by adding only the 'other' constraints along with random specific constraints (5% of the grid in my case). This is essential because the SAT solver tends to generate the same Sudoku without some form of randomization in the constraints.

Once we have a fully solved Sudoku, we start by creating a base.cnf file. This file includes all the generic constraints and the negation of the solution (Other Constraints). The advantage of using a base.cnf file is that it eliminates the need to rewrite the entire file during each iteration, significantly improving computation times, especially for larger Sudoku puzzles, such as the 25x25 version.

In each iteration, we copy the base.cnf file, remove a clue, and then write the specific constraints to determine if the Sudoku still has a unique solution. If the removal of a clue results in multiple solutions, we reinstate the clue and attempt to remove a different one.

For 9x9 and 16x16 Sudoku puzzles, my goal is to remove approximately 60% of the clues. However, for the 25x25 Sudoku, I limit the process to 300 iterations due to computational time constraints.

I was able to generate 9x9, 16x16 and 25x25 sudokus in fair amount of time (3 min approximately for the 25x25 sudoku)

To add the possibility of creating a Sudoku with N-1 digits, we begin by randomly removing all the clues of the same digit. Then, we repeat the previous steps.

This method allows us to generate a 9x9 Sudoku using only 8 digits, which can be achieved using the -cm argument.