Sudoku

Domain:
This problem domain consists of 9x9 sudoku grids composed of 81 cells. Some cells are pre filled with digits, and the goal is to fill all empty cells so that every row, column, and 3x3 subgrid contains all of the digits from 1 to 9 exactly once. The variables are the 81 cells on the grid, and each variable has the domain of possible values, the digits 1 through 9. The constraints are the rules listed above, that no digit can be repeated in any row, column, or 3x3 subgrid.

The problem domain also includes a simpler 4x4 grid that follows the same rules, but with digits 1 through 4 and 2x2 subgrids.

Formalization <X, D, C>:
X (Variables):
Each cell is a variable that is represented by an alphanumeric code. They all start with V for variable, then a column indicator followed by a row indicator. For example, Vb5 represents the cell located in column 2, row 5. Similarly, Vg1 is the cell located in column 7, row 1.

D (Domains):

```
# Row 1
  Va1: [1,2,3,4,5,6,7,8,9]
  Vb1: [1,2,3,4,5,6,7,8,9]
  Vc1: [1,2,3,4,5,6,7,8,9]

  Vd1: [1,2,3,4,5,6,7,8,9]
  Ve1: [1,2,3,4,5,6,7,8,9]
  Vf1: [1,2,3,4,5,6,7,8,9]

  Vg1: [6]
  Vh1: [1,2,3,4,5,6,7,8,9]
  Vi1: [1,2,3,4,5,6,7,8,9]
```

If the cell is empty, it has the entire domain of digits available to it, represented by [1,2,3,4,5,6,7,8,9]. If the cell is filled, it only has one digit available, represented like so: [6].

C (Constraints):

```
# Row constraints
  alldiff(Va1, Vb1, Vc1, Vd1, Ve1, Vf1, Vg1, Vh1, Vi1)
  alldiff(Va2, Vb2, Vc2, Vd2, Ve2, Vf2, Vg2, Vh2, Vi2)
  alldiff(Va3, Vb3, Vc3, Vd3, Ve3, Vf3, Vg3, Vh3, Vi3)
  alldiff(Va4, Vb4, Vc4, Vd4, Ve4, Vf4, Vg4, Vh4, Vi4)
  alldiff(Va5, Vb5, Vc5, Vd5, Ve5, Vf5, Vg5, Vh5, Vi5)
  alldiff(Va6, Vb6, Vc6, Vd6, Ve6, Vf6, Vg6, Vh6, Vi6)
  alldiff(Va7, Vb7, Vc7, Vd7, Ve7, Vf7, Vg7, Vh7, Vi7)
  alldiff(Va8, Vb8, Vc8, Vd8, Ve8, Vf8, Vg8, Vh8, Vi8)
  alldiff(Va9, Vb9, Vc9, Vd9, Ve9, Vf9, Vg9, Vh9, Vi9)

  # Column constraints
  alldiff(Va1, Va2, Va3, Va4, Va5, Va6, Va7, Va8, Va9)
  alldiff(Vb1, Vb2, Vb3, Vb4, Vb5, Vb6, Vb7, Vb8, Vb9)
  alldiff(Vc1, Vc2, Vc3, Vc4, Vc5, Vc6, Vc7, Vc8, Vc9)
  alldiff(Vd1, Vd2, Vd3, Vd4, Vd5, Vd6, Vd7, Vd8, Vd9)
  alldiff(Ve1, Ve2, Ve3, Ve4, Ve5, Ve6, Ve7, Ve8, Ve9)
  alldiff(Vf1, Vf2, Vf3, Vf4, Vf5, Vf6, Vf7, Vf8, Vf9)
  alldiff(Vg1, Vg2, Vg3, Vg4, Vg5, Vg6, Vg7, Vg8, Vg9)
  alldiff(Vh1, Vh2, Vh3, Vh4, Vh5, Vh6, Vh7, Vh8, Vh9)
  alldiff(Vi1, Vi2, Vi3, Vi4, Vi5, Vi6, Vi7, Vi8, Vi9)

  # 3x3 box constraints
  alldiff(Va1, Vb1, Vc1, Va2, Vb2, Vc2, Va3, Vb3, Vc3) # top left
  alldiff(Vd1, Ve1, Vf1, Vd2, Ve2, Vf2, Vd3, Ve3, Vf3) # top middle
  alldiff(Vg1, Vh1, Vi1, Vg2, Vh2, Vi2, Vg3, Vh3, Vi3) # top right
  alldiff(Va4, Vb4, Vc4, Va5, Vb5, Vc5, Va6, Vb6, Vc6) # middle left
  alldiff(Vd4, Ve4, Vf4, Vd5, Ve5, Vf5, Vd6, Ve6, Vf6) # middle middle (center)
  alldiff(Vg4, Vh4, Vi4, Vg5, Vh5, Vi5, Vg6, Vh6, Vi6) # middle right
  alldiff(Va7, Vb7, Vc7, Va8, Vb8, Vc8, Va9, Vb9, Vc9) # bottom left
  alldiff(Vd7, Ve7, Vf7, Vd8, Ve8, Vf8, Vd9, Ve9, Vf9) # bottom middle
  alldiff(Vg7, Vh7, Vi7, Vg8, Vh8, Vi8, Vg9, Vh9, Vi9) # bottom right
```

Note: The formalization is the same for the 4x4 grid, just only extending to the smaller domain and utilizing 2x2 subgrids.

Heuristic:
I implemented the Minimum Remaining Values heuristic. The function of this heuristic is to allow the solver to choose the cells that have the fewest available options that could possibly be assigned to solve first. It prioritizes the cells under with the most constraints, which are the ones with the fewest possible options from the domain. It is in this way that it fills cells quicker and effectively reduces the domains of other cells in the same row, column, and subgrid, allowing the search tree to be pruned much quicker than the base solver.

Reflection:
To test the performance of the solver, I opted to count the number of steps taken to reach the solution. A step is defined as a tentative assignment of a variable. Below are the metrics for the base solver and the solver with the MRV heuristic added.

Base:

- Instance 1 (4x4): 31 steps
- Instance 2 (9x9 medium): 25,167 steps
- Instance 3 (9x9 hard): 169,810 steps

MRV:

- Instance 1 (4x4): 16 steps
- Instance 2 (9x9 medium): 81 steps
- Instance 3 (9x9 hard): 311 steps

It becomes extraordinarily clear how the addition of the heuristic exponentially reduces the amount of work it takes to solve each instance of the problem. The MRV heuristic works extremely well for the sudoku problem and its constraints, it seems to be very well suited for this type of grid based problem. The amount of guessing is completely slashed, allowing the solver to form a solution much quicker than without the heuristic. The base solver just makes arbitrary "guesses" that lead to very deep, incorrect search paths before backtracking. This does not happen nearly as much with the MRV heuristic in place.

Git Repository:
https://github.com/joeoneal/cs4300/tree/8a91d42173d82f79f1d179c573e4792e1fae8917/cs4300-csp