CS 4300: Assignment #3 - Constraint Satisfaction Problem

Brandon Lewis

Date: September 17, 2025

Problem Domain: 9x9 Sudoku

GitHub Repository

The source code for the modified solver and the .csp instance files can be found at the following repository:

https://github.com/BranPLewis/Sudoku_9x9

---

1. Problem Description: Sudoku

Sudoku is a logic-based number-placement puzzle. The objective is to fill a **9×9 grid** with digits so that each column, each row, and each of the nine **3×3 subgrids** that compose the grid contain all of the digits from 1 to 9. The puzzle starts with a partially completed grid, which for a well-posed puzzle has a single unique solution.

The challenge is to fill in the remaining empty cells while adhering to these three core constraints simultaneously:

1. Each row must contain the numbers 1 through 9, without repetition.
2. Each column must contain the numbers 1 through 9, without repetition.
3. Each of the nine 3×3 subgrids must contain the numbers 1 through 9, without repetition.

---

2. Formalization as <X, D, C>

A Sudoku puzzle can be formally modeled as a Constraint Satisfaction Problem (CSP) using the <X, D, C> framework.

- X: Variables
  The set of variables, X, consists of 81 variables, one for each cell in the 9x9 grid. We can denote each variable as V_rc, where r is the row index ($1 \le r \le 9$) and c is the column index ($1 \le c \le 9$).
  $X=\{V_{rc} \mid 1 \le r \le 9, 1 \le c \le 9\}$
- D: Domains
  The domain, D, for each variable is the set of possible values it can take. For any empty cell in the grid, the initial domain is the set of all possible digits.
  $D_{rc}=\{1,2,3,4,5,6,7,8,9\}$
  For cells that are pre-filled in a specific puzzle instance with a value k, the domain for that variable is restricted to just that single value, e.g., D_rc=k.

- C: Constraints
  The constraints, C, ensure that the rules of Sudoku are followed. These are all alldifferent constraints applied to three different types of groups.
    1. Row Constraints: For each row r, all variables in that row must be unique.
       alldiff(Vr1,Vr2,…,Vr9)for each r∈{1,…,9}
    2. Column Constraints: For each column c, all variables in that column must be unique.
       alldiff(V1c,V2c,…,V9c)for each c∈{1,…,9}
    3. 3x3 Subgrid Constraints: For each of the nine 3x3 subgrids, all variables within that subgrid must be unique. For example, the top-left subgrid constraint is:
       alldiff(V11,V12,V13,V21,V22,V23,V31,V32,V33)
       There are a total of 9 row, 9 column, and 9 subgrid constraints, making 27 alldiff constraints in total.

---

3. Heuristic Implementation: Minimum Remaining Values (MRV)

To extend the backtracking solver, I implemented the **Minimum Remaining Values (MRV)** heuristic. This is a variable-ordering heuristic that improves the efficiency of the search process.

- **How it Works:** Instead of selecting the next unassigned variable in a fixed, static order, the MRV heuristic dynamically chooses the variable with the **fewest legal values** remaining in its domain.
- **Why it is Effective:** This strategy is often called the "fail-first" principle. By selecting the most constrained variable, we are more likely to identify a dead-end in the search tree early on. For example, if a variable's domain has been reduced to a single possible value, it is critical to test that assignment immediately. If it leads to a contradiction, we can prune that entire branch of the search tree without wasting time exploring other variable assignments. In Sudoku, this perfectly mimics the human approach of finding a cell where only one number can possibly go.

4. Results and Reflection

This section presents the solutions found by the solver for the `.csp` instances and analyzes the performance improvement gained by using the MRV heuristic.

Solution for sudoku_1.csp

Without heuristic
Steps it took: 1113
Time it took: 1.58 seconds
Solution:

1 6 3 | 9 4 8 | 5 2 7
7 8 2 | 6 5 1 | 3 9 4
4 9 5 | 2 7 3 | 1 8 6
—-----------------------
6 7 9 | 3 1 2 | 4 5 8
3 4 8 | 5 9 7 | 6 1 2
2 5 1 | 8 6 4 | 7 3 9
—-----------------------
8 2 6 | 7 3 5 | 9 4 1
9 3 4 | 1 2 6 | 8 7 5
5 1 7 | 4 8 9 | 2 6 3

With heuristic
Heuristic: MRV (Minimum Remaining Values)
Steps it took: 95
Time it took: 0.026 seconds
Solution:

1 6 3 | 9 4 8 | 5 2 7
7 8 2 | 6 5 1 | 3 9 4
4 9 5 | 2 7 3 | 1 8 6
----
6 7 9 | 3 1 2 | 4 5 8
3 5 8 | 7 9 4 | 6 1 2
2 4 1 | 8 6 5 | 7 3 9
----
8 2 6 | 5 3 7 | 9 4 1
9 3 4 | 1 2 6 | 8 7 5
5 1 7 | 4 8 9 | 2 6 3

Solution for sudoku_2.csp

Without heuristic
Steps it took: 344
Time it took: 0.16 seconds
Solution:

6 8 2 | 3 5 4 | 9 1 7
4 1 9 | 6 8 7 | 3 5 2
7 3 5 | 1 9 2 | 8 6 4
----
3 9 6 | 2 7 5 | 4 8 1
1 2 8 | 4 3 6 | 5 7 9
5 7 4 | 8 1 9 | 6 2 3
----
2 5 7 | 9 6 3 | 1 4 8
9 6 1 | 7 4 8 | 2 3 5
8 4 3 | 5 2 1 | 7 9 6

With heuristic
Heuristic: MRV (Minimum Remaining Values)
Steps it took: 120
Time it took: 0.028 seconds
Solution:

6 8 2 | 3 5 4 | 9 1 7
4 1 9 | 6 8 7 | 3 5 2
7 3 5 | 1 9 2 | 8 6 4
----
3 9 6 | 2 7 5 | 4 8 1
1 2 4 | 8 6 3 | 5 7 9
5 7 8 | 4 1 9 | 6 2 3
----
2 5 7 | 9 3 6 | 1 4 8
9 6 1 | 7 4 8 | 2 3 5
8 4 3 | 5 2 1 | 7 9 6

Solution for sudoku_3.csp

Without heuristic
Steps it took: 6156
Time it took: 1.97 seconds
Solution:

```
5 3 4 | 6 7 8 | 9 1 2
6 7 2 | 1 9 5 | 3 4 8
1 9 8 | 3 4 2 | 5 6 7
-----------------------
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
-----------------------
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9
```

With heuristic
Heuristic: MRV (Minimum Remaining Values)
Steps it took: 81
Time it took: 0.019 seconds
Solution:

```
5 3 4 | 6 7 8 | 9 1 2
6 7 2 | 1 9 5 | 3 4 8
1 9 8 | 3 4 2 | 5 6 7
-----------------------
8 5 9 | 7 6 1 | 4 2 3
4 2 6 | 8 5 3 | 7 9 1
7 1 3 | 9 2 4 | 8 5 6
-----------------------
9 6 1 | 5 3 7 | 2 8 4
2 8 7 | 4 1 9 | 6 3 5
3 4 5 | 2 8 6 | 1 7 9
```

Analysis

What I found is that Minimum Remaining Values in the terms of a 9x9 sudoku board was crucial in finding rows and columns and 3x3 sub-grids that had few numbers left to fill, forcing the solver to either fill them successfully or find a bad path in need of backtracking. I noticed a significant reduction in the amount of steps with the solver using the MRV heuristic. I created 3 instances of a 9x9 sudoku board, each of which has a different increasing number of starting tiles, and a different number of solutions occurred for each. Each instance MRV found the solution in roughly ⅓ less steps then without MRV.