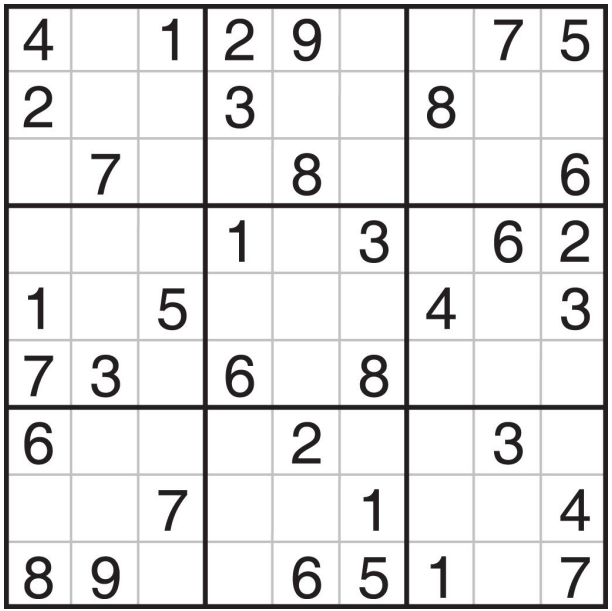
Sudoku Solver with Backtracking and CSP Techniques

By: Justin Guidry, Jonas Long
Course: Intro to AI 331
Rochester Institute of Technology, GCCIS
Date: December 3rd 2025

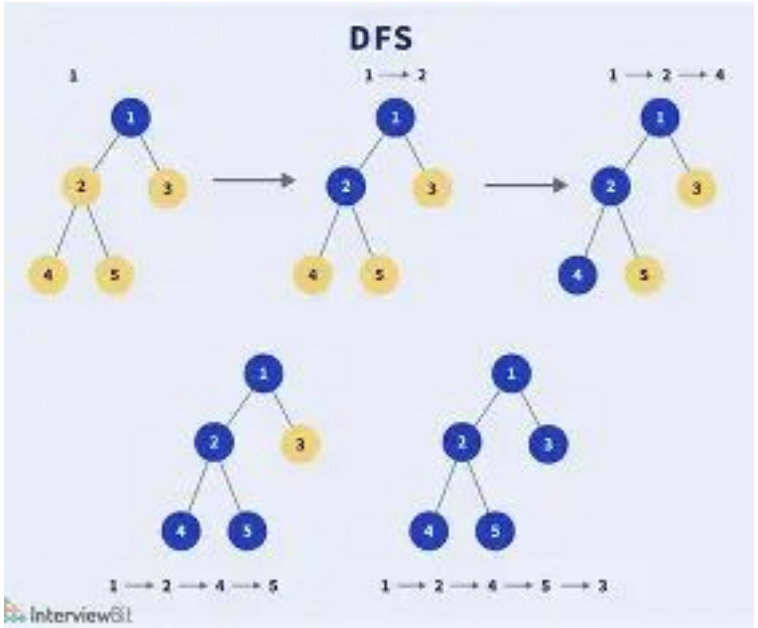
What is Sudoku

- Puzzle where you need to fill in a 9x9 board with numbers 1-9
- Each cell must have a unique number within its column, row, and in the 3x3 area it sits in
- Boards start pre-filled, usually with enough cells filled so that there is only 1 possible solution
- Solving the board is an exercise in logic- making educated guesses and ruling out possibilities until the remaining cells only have one option left



What is DFS

- DFS is a graph search which makes use of a stack data structure in order to process nodes for exploration (LIFO behavior)
- Once it is done with a path it will backtrack to the next path and repeat the process until it either explores all nodes or finds a goal node
- This is how we implement backtracking for Sudoku



Constraint Satisfaction Problems (CSP)

A mathematical problem where variables must be assigned a value in their domain such that it meets all given constraints

- Variables: These are what we need to find values for (each territory)
- Domains: The set of possible values for a given variable (Colors Red, Green, Blue)
- Constraints: The rules that dictate what values are allowed for a given variable (If two territories touch then they can't have the same color)

We will model Sudoku as a CSP in order to improve backtracking

Basic Backtracking vs CSP Backtracking

Hypothesis

- The basic backtracking should explore more states since it is doing no pruning on the initial domains of all the variables
- Using forward checking in the backtracking will result in fewer possible states since when exploring a branch the domains of neighboring states will be pruned down so there will be less to explore

Methods

- Solving Sudoku with DFS
  - Try every possible board using DFS
- Solving Sudoku with DFS and simple check
  - Before placing a value check if it is safe to place
- Solving Sudoku with DFS enhanced with forward checking
  - Treat the puzzle as a CSP and use forward checking in order to process less branches

Sudoku with DFS

State space: The starting board and all board states made from filling in the blank spaces in the start board

- Successor function: Assign a value 1-9 to an empty space
- Start state: The starting board, which has one or more spaces filled in
- Goal test: The current board is completely filled in, and each space is a different number than those in its column, row, and group

A sample starting board

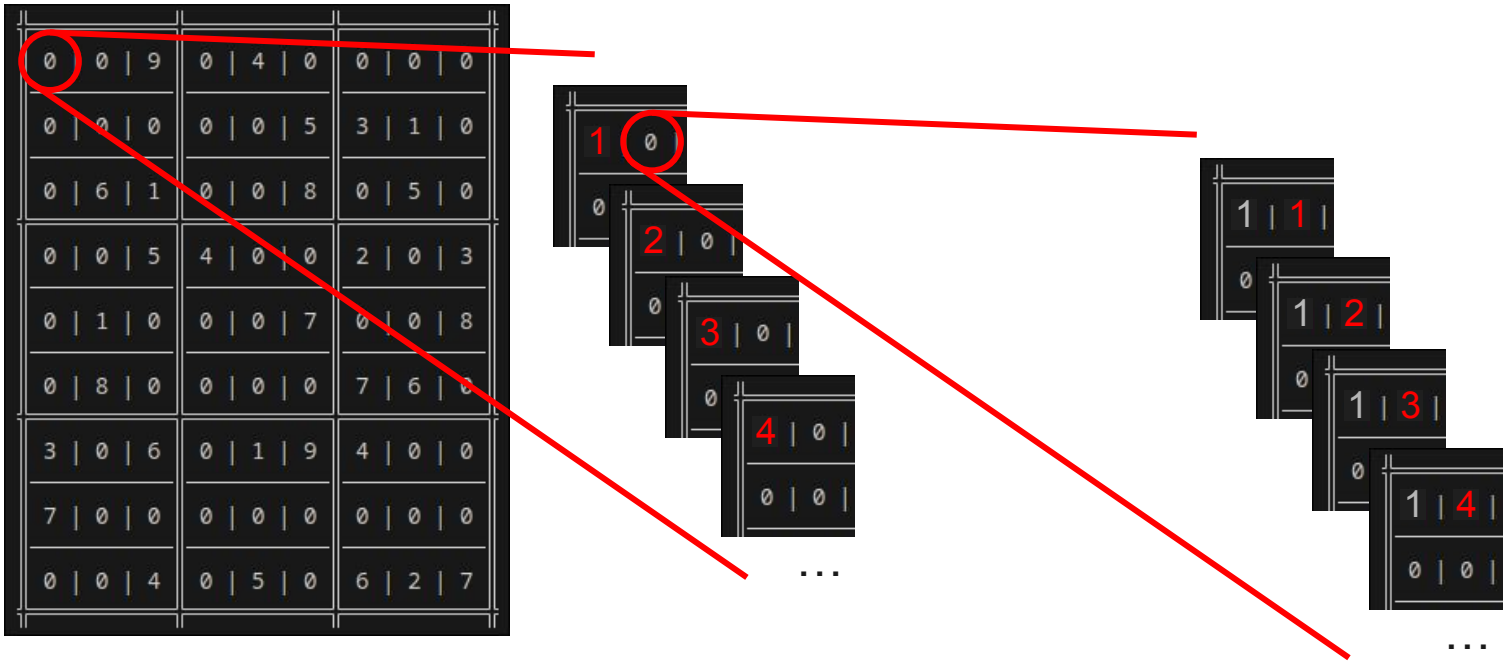
0s are empty spaces

0	0	9	0	4	0	0	0	0
0	0	0	0	0	5	3	1	0
0	6	1	0	0	8	0	5	0
0	0	5	4	0	0	2	0	3
0	1	0	0	0	7	0	0	8
0	8	0	0	0	0	7	6	0
3	0	6	0	1	9	4	0	0
7	0	0	0	0	0	0	0	0
0	0	4	0	5	0	6	2	7

One possible solution

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

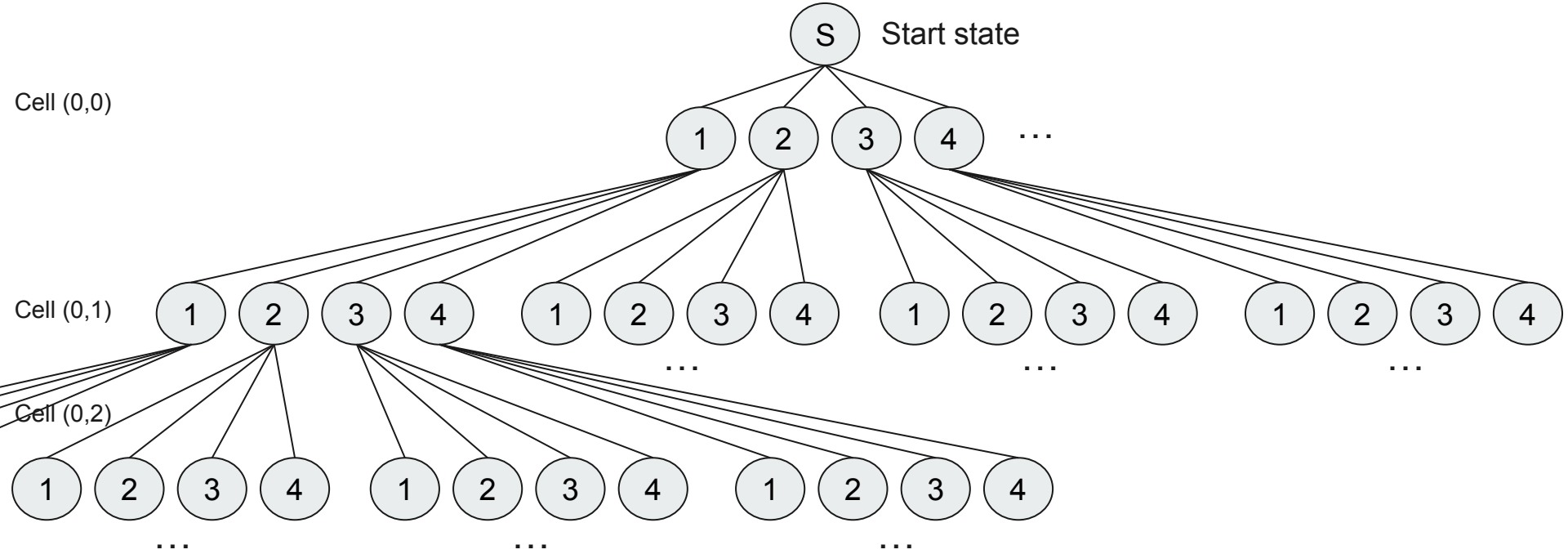
Solving Steps



As a Search Tree

Given this initial row 0:

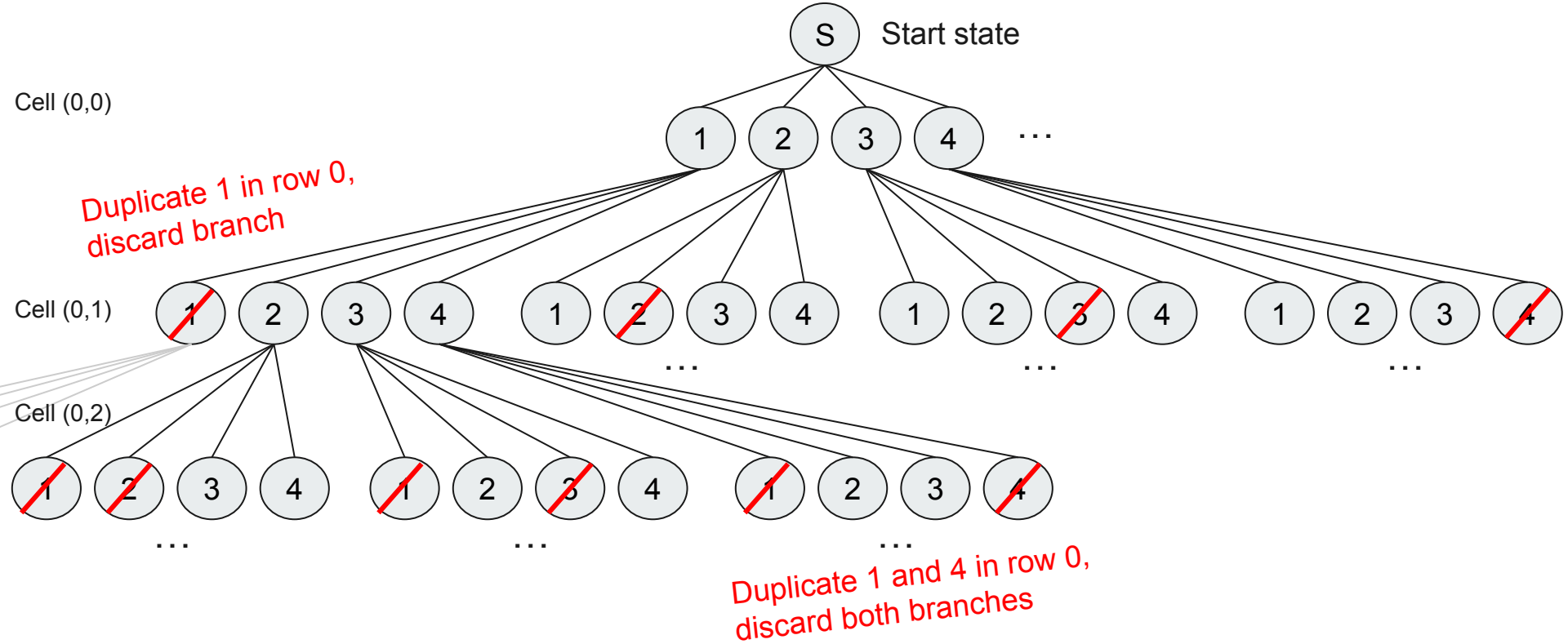
0	0	9	0	7	0	0	0	0



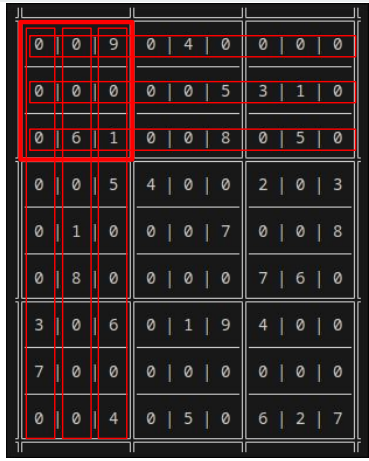
Given this initial row 0:

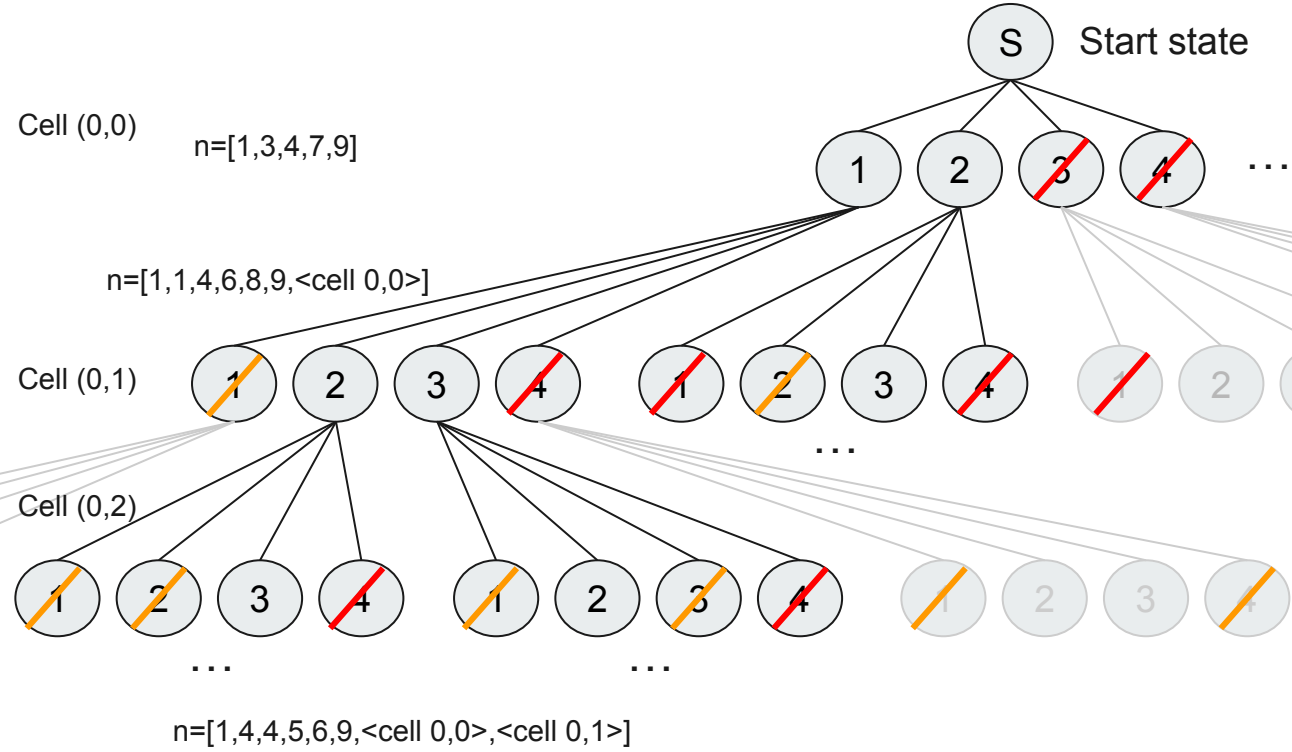
0	0	9	0	7	0	0	0	0

With CSP Backtracking



More Pruning





When considering column, row, and group neighbors, pruning is even more thorough

Backtracking in Sudoku

At each step, check that the assigned value will be valid in the grid

Saves time evaluating results for a grid that will never be possible (e.g. grid where we assign 1s to every single empty space)

Test:

Do any column, row, or group neighbors have the same value? If so, abandon this branch of the search tree

Backtracking Cont.

No need to deduplicate nodes- each DFS node is a unique state

No inherent advantage to using BFS- solution depth is constant

Using DFS optimizes memory usage

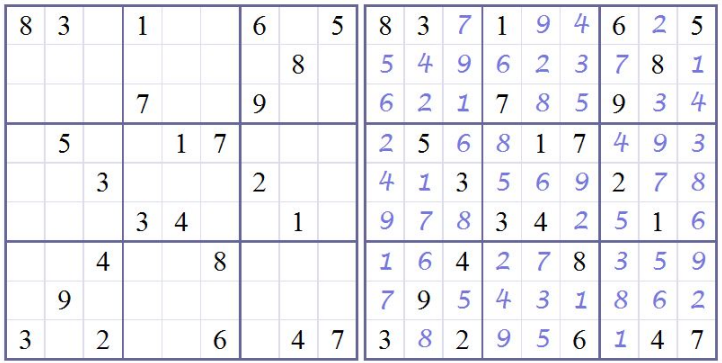
Pruning is performed on branches that will never produce valid board states-

backtracking solution is the same as the naive solution

Sudoku as a CSP

We framed Sudoku as a Constraint Satisfaction Problem

- Identification problem
- High-order constraints (81 variables, 1-9 is initial domain)



- Start with a partial assignment
- Build complete assignments
- Eventually settle on a solution, a complete assignment that meets constraints

Forward checking in Sudoku

For every empty cell:

- Assign each empty cell a domain list, with the numbers 1-9
- Assign the pre-placed value to the domain of cells that are pre-filled

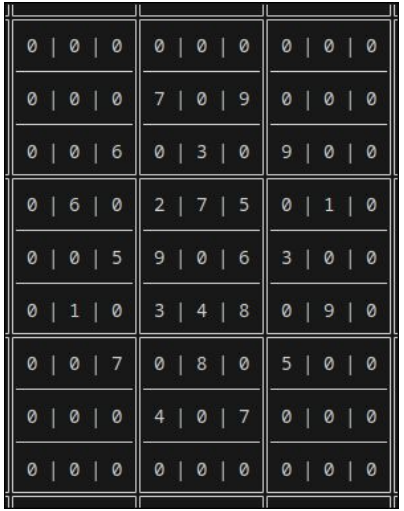
Every recursion:

- Update domain list by removing numbers used by neighboring cells from the domain list
- Recurse on every possible cell value for the domain

Results

- Plain DFS
  - Never finished (57 0's = 6.3*10^15 branches)
- DFS with backtracking
  - Solve time: 32.5s on average
  - Total branches explored: 535051
- DFS with backtracking and forward checking
  - Solve time: 17s on average
  - Total branches explored: 436329

- Optimization effects with forward checking:
  - 15.5s reduction in search time
  - 18.5% reduction in branch exploration



Stress test with mostly-blank board on a laptop

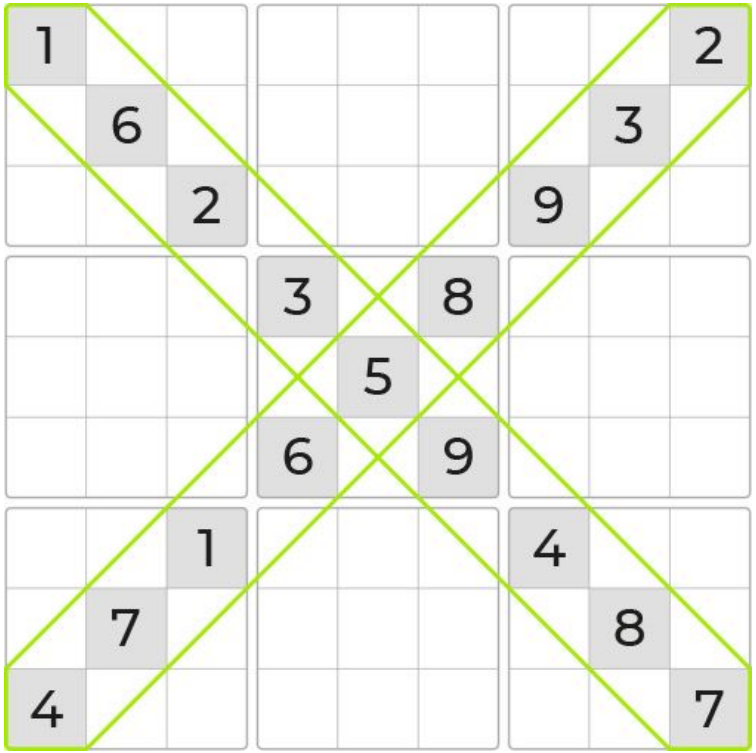
Conclusions

- Sudoku can be feasibly treated as a tree search problem
- The solver can take advantage of Constraint Satisfaction Problem solving techniques for further optimization

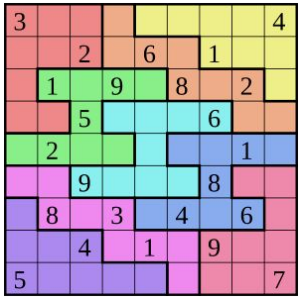
- More optimizations to consider:
  - Could cache neighbor list to reduce list access (memory vs cpu)
  - Would be orders of magnitude faster in a compiled language
  - Domain updates can be cached and made deterministic based on only earlier cell updates

Extension: Diagonals

- Along with the base rules for Sudoku we implemented another custom rule for the diagonals
- This is where along both main diagonals you can only have exactly one of each number 1 to N
- Usually referred to as X Sudoku

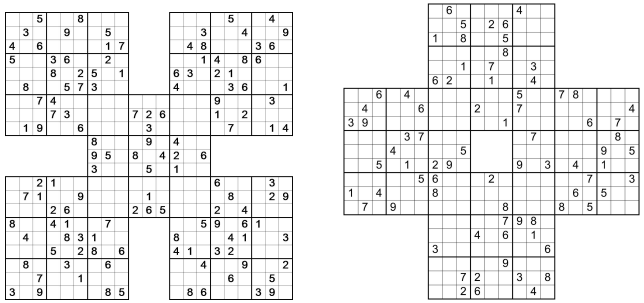


Further Research



Jigsaw Sudoku

Giant Sudoku boards



- Solving for more advanced board shapes
  - Would require a more complex structure for storing the board state and figuring out which areas are connected
  - CSP solution methodologies would remain largely unchanged
- More complicated solution criteria
  - Requires more complicated solution tests
  - Backtracking and forward checking needs reimagining

Killer Sudoku, before and after solving

References

- Course materials on CSP
- https://en.wikipedia.org/wiki/Sudoku
- https://prntbl.concejomunicipaldechinu.gov.co/sudoku-printables-6-per-page/
- https://www.geeksforgeeks.org/artificial-intelligence/constraint-satisfaction-problems-csp-in-artificial-intelligence/