AI Sudoku Solver

By: Peter, Patrick, Simar, Sepehr

Overview

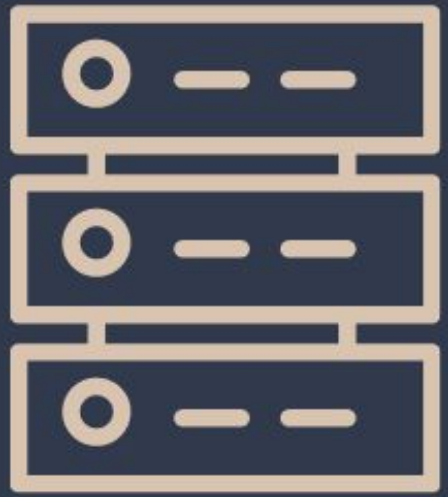
- Tools/Languages
- Input File Generation
- Table Results
- Brute Force Heuristics
- CSP Heuristics
- Demo

Tools/Languages

Front-end

- HTML - Base elements in UI
- CSS - Sudoku Styling
- JavaScript - Board Creation
- Bootstrap - Modem

Back-end



- Python
- Fast API
- Server-sent events
- Azure Virtual Machine
- Azure Storage

Input File Generation

Creation:

1. Creates a clean board with appropriate dimensions
2. Creates sets for rows, columns and, sub squares
3. Fills in values for unassigned cells 25% of the time
    a. Checks sets to see if the value is valid, if not try another value
4. Keep repeating step 3 until 25% of the board is filled

Selection (currently used):

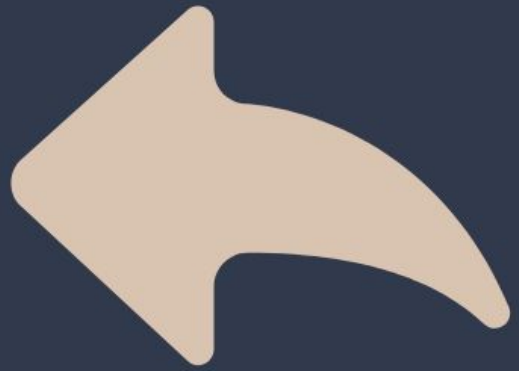
1. Solved board randomly selected in pool by size
2. Masked 75% of the board to ‘0s’

Table Results

Size	Average Time	Standard Deviation
9x9	0.073s	0.144s
12x12	0.331s	0.598s
16x16	2.95s	0.598s
25x25 (5 Solved)	61.7s	51.7s

Brute Force Heuristics

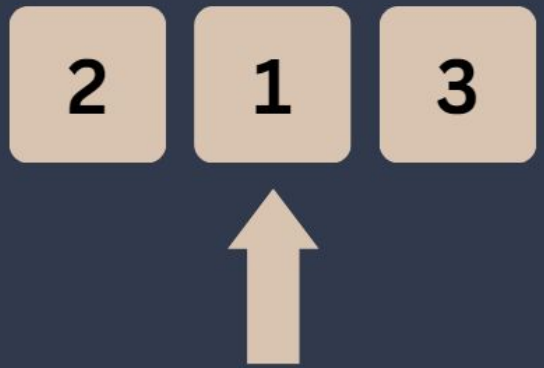
Iterative Backtrack



1. Checks if the current node is a solution
2. If not, generates child nodes representing possibilities for filling the next empty cell
3. Checks for unexplored child nodes pushes to stack
4. When there are no unexplored child nodes, the current node is marked and removed from the stack. The search backtracks to the previous node.

Implements a timeout mechanism where if the stack is cleared (excluding root node) if there is no change after 10000 iterations.

Prioritize Minimum Domain



1. Get the domains for each cell on the board

2. Assign a heuristic value to each cell based on the size of it’s domain

3. Select the cell with the smallest heuristic value to fill in next

Tie-breaker

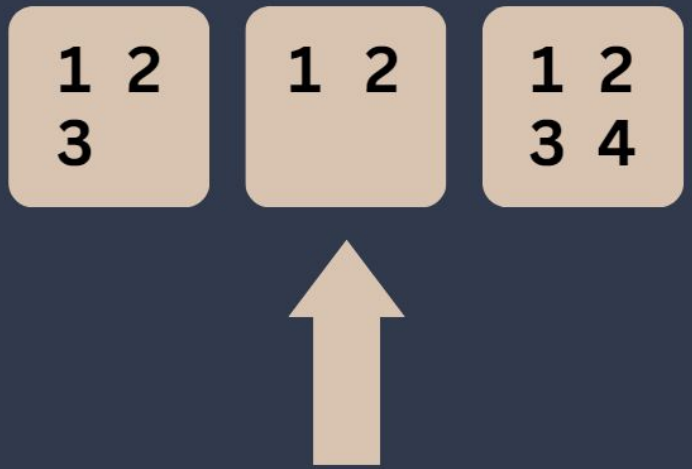
- Tiebreak by selecting empty cells with fewer assigned neighbors.
- Such cells have more freedom to accommodate different values and increase the chance of finding a valid solution.
- Prioritizing cells that are more likely to yield a correct solution increases the solver's efficiency.

Reserved Stack

- Problem: Get stuck in a local optimal
- Custom timeout counter to periodically migrate nodes to a reserved stacks
- Escape from local optimal, while ensuring nodes could be preserved for future evaluation

CSP Heuristics

MRV/Degree



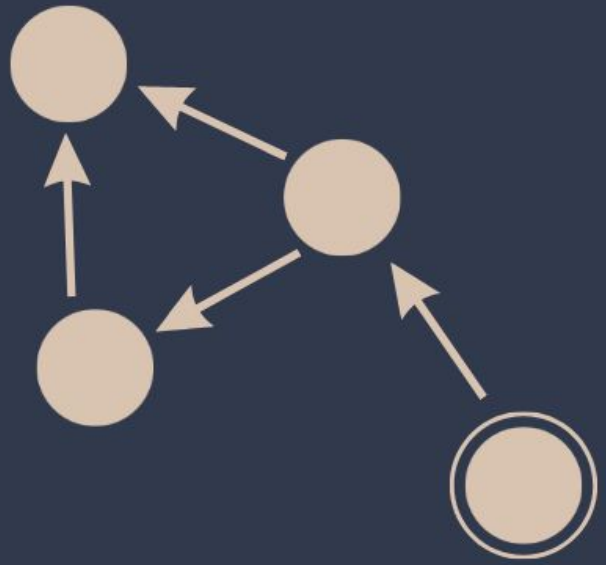
MRV (Applied 1st):

- Selects the cell with the fewest legal values
- Reduces the search space, minimizing likelihood of backtracking

Degree (Tie-breaker):

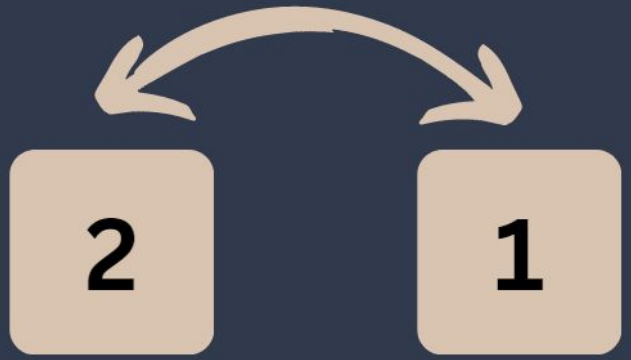
- Selects the cell involved with the highest number of unassigned cells
- Prioritizes the filling of cells that in-turn constrain more unassigned cells

LCV



- To determine the order of values, we use the least constraining value heuristic.
- This heuristic assigns a value that has the least impact on its neighboring cells.
- By choosing a value that eliminates the fewest options for other variables, this heuristic minimizes backtracking and increases the algorithm's efficiency.

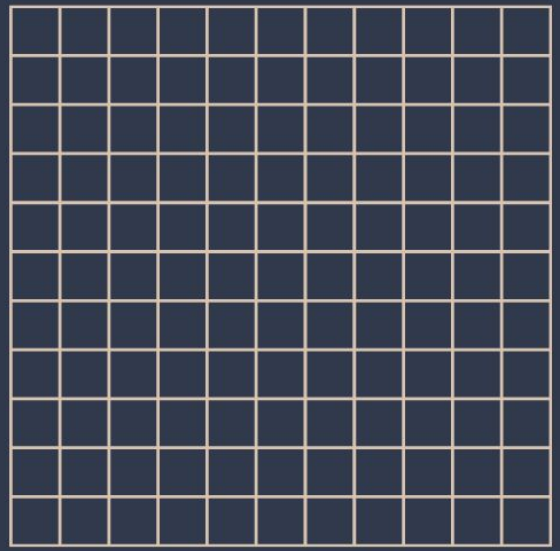
MAC (AC-3)



Checks whether domain values of each cell are consistent:

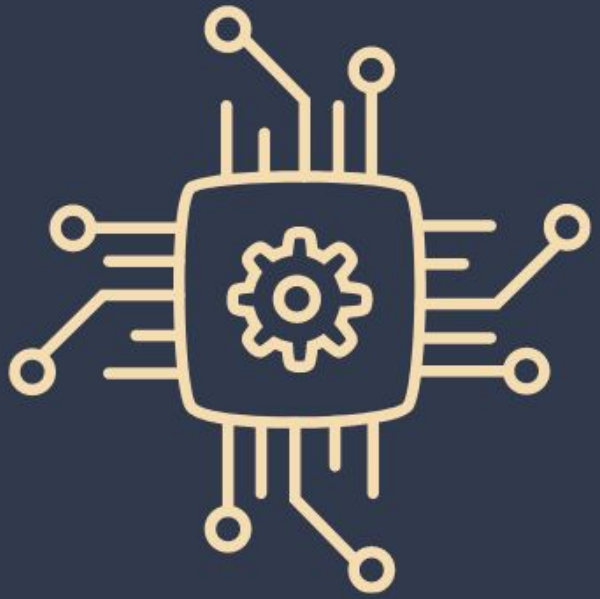
- If consistent, the branch of that node is expanded and the search continues
- If inconsistent, the node is flagged as checked and popped off the stack for backtracking

Sudoku Specific Rules



- Hidden Single Rule
- Naked Pair Rule

Multi-processing



1. Expand the first node
2. Run the algorithm on each of the child nodes in parallel
3. Return solution when found
4. Terminate other processes

Demo Time