CS3243 Project 2: Solving Sudoku Puzzles as Constraint Satisfaction Problems (CSPs)

Cheng Lit Yaw A0185639M, Jonathan Tjendana A0185549M,
Wang Zicong A0184475U, and Yang Zi Yun A0184682U

National University of Singapore, 21 Lower Kent Ridge Rd, Singapore 119077

1 Problem definition

Table 1. Problem Definition for Sudoku Puzzles as CSP

Variables	Each of the 81 squares is a variable.
Domains	Initial domain for each variable is all the possible values: [1, 2, 3, 4, 5, 6, 7, 8, 9].
Constrains	Each variable in the same column, same row or same box cannot have the same value. It is represented in binary constraints: Vrc ≠ Vr'c' for all the pairs {rc, r'c'} such that rc and r'c' are in the same column, same row or same box.
State	Each state is a partial assignment of values to variables. Initial state is an empty assignment.
Action	Assign a valid value to an unassigned variable, fail if no valid assignments.
Transition Model	Generate state with the next variable assigned. T(current state $S_{current}$, action) = {next state S_{new}}
Goal Test	Complete assignment of values to all variables that is consistent with all the constraints.
Cost Function	1 for each step.

2 CSP Algorithms/Heuristics Description and Analysis

2.1 Variable ordering mechanisms/heuristics: Minimum Remaining Values (MRV) vs without MRV

We choose to implement MRV heuristic. We would first assign variables with the domain with the least remaining values. This is the variable that is most likely to cause failure in a branch. This will prune the bigger branches and reduce the number of unnecessary nodes being expanded during backtracking, making the search faster.

2.2 Value ordering heuristics: Least Constraining Value (LCV)

Least Constraining Value is chosen. For each variable to be assigned, we choose its value that eliminates the fewest choices from neighbouring variables. This keeps maximum flexibility for remaining variable assignments and helps to avoid failure by choosing the branch that has a higher probability to succeed, thus backtracking would happen less often.

2.3 Inference mechanisms: Forward Checking (FC) vs Arc Consistency (AC-3)

Both FC and AC3 detect failure for a particular assignment, thus pruning out invalid branches. Although AC-3 provides a further constraint propagation and may detect failure earlier than FC, we decide to choose FC over AC-3. This is because AC-3 has a time complexity of $O(n^2d^3)$ but less visited nodes, while FC has a complexity of $O(nd)$ but more visited nodes. After considering the trade-off of number of nodes and the processing time for each node, we decided that FC would be faster and sufficient enough for a sudoku solver in general.

Hence, the final choice for our sudoku solver is: backtracking algorithm with MRV as variable ordering heuristic, LCV as value ordering heuristic and FC as inference mechanism.

3 Experimental setup

3.1 Goal of the experiment

- To determine whether MRV heuristic will increase the performance of sudoku solver, in order to justify our analysis in Section 2.1

3.2 Experimental Setup

- Input: 19 different sudoku puzzles. The input puzzles can be found in the folder named *experiment_cases/* (Source: https://github.com/t-dillon/tdoku and http://norvig.com/sudoku.html)
- Output: number of visited nodes and time taken to solve the puzzle
- The puzzles were solved using Backtracking with Least Constraining Values and Forward Checking, the only difference is the presence of MRV heuristics
- The code with MRV is named *MRV.py*, and the code without MRV is named *noMRV.py* in our submission folder

3.3 Experiment Results

- Table 2 below shows the partial experiment result with representative inputs. Please refer to Appendix A for the full experiment result.

Table 2. Number of visited nodes and time taken (Partial Version)

Inputs	method with MRV		method without MRV	
	Number of visited nodes	Time taken	Number of visited nodes	Time taken
1	47	0.026752	47	0.024959
6	58	0.036817	257	0.157851
7	64	0.040332	1177	0.681736
8	100	0.066645	2470	1.429404
12	170	0.111091	396	0.232178
13	375	0.240020	1782	1.085160
15	733	0.431736	2597	1.341955
17	1107	0.691667	1373	0.831356
18	1239	0.762580	11808	6.481448
19	1731	1.112580	5547	3.036656

4 Results, Analysis and Rationale for Selection

4.1 Analysis and Rationale

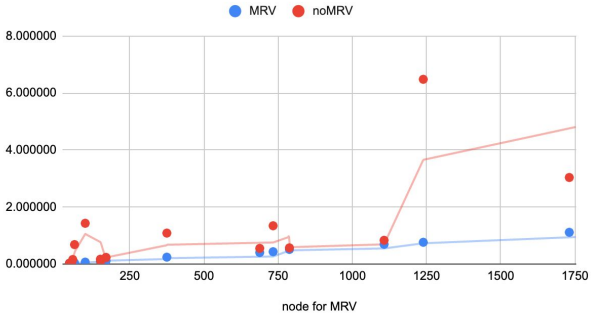

Fig.1. Time taken vs number of visited nodes by MRV

Our solutions are able to complete all possible puzzles in our input test cases. Fig.1 shows the time consumed for the two methods to run. The x-axis is the number of nodes traversed by the MRV method (which is taken as an indicator for the difficulty of the puzzle), and the y-axis is the time ran. The trendlines are the moving average time of the two methods.

From Table 2, we can see that the solution with MRV outperforms the solution without MRV after a certain difficulty. We believe that the factors affecting the speed of solving sudoku puzzles are the number of visited nodes (which represents how extensively the backtracking search goes) and the time consumed in each node.

At a higher level of difficulty, due to MRV reducing the number of nodes being expanded, the code is able to run within less time to solve the sudoku. When doing extremely easy sudoku where the number of nodes is small, the solution without MRV will outperform the MRV heuristic. This is because both methods run through a similar number of nodes, but the method with MRV consumes a bit more overhead time to run the MRV algorithm in each node. However, while the difficulty increases, as shown in Fig.1, the method with MRV outperforms the method without MRV, as it runs significantly lesser nodes.

For the easy puzzles, although MRV takes a slightly longer time, it still completes the puzzle efficiently enough, hence the longer time is negligible. For the difficult puzzles, MRV provides a significant improvement in the time taken as compared to the solution without MRV. Taking all levels of difficulties of puzzles into account, we would conclude that MRV heuristic is a better solution than the one without MRV in general, which supports our statement in Section 2.1.

Appendix A. Complete Experiment Result

Table 3. Number of visited nodes and time taken (Complete Version)

Inputs	MRV		without MRV	
	Number of visited nodes	Time taken	Number of visited nodes	Time taken
1	47	0.026752	47	0.024959
2	47	0.028504	47	0.024777
3	49	0.027726	49	0.025936
4	51	0.028985	51	0.027321
5	54	0.031520	53	0.029803
6	58	0.036817	257	0.157851
7	64	0.040332	1177	0.681736
8	100	0.066645	2470	1.429404
9	151	0.095206	271	0.168584
10	152	0.095060	178	0.099072
11	170	0.119923	396	0.225560
12	170	0.111091	396	0.232178
13	375	0.240020	1782	1.085160
14	688	0.402607	929	0.548725
15	733	0.431736	2597	1.341955
16	788	0.511490	896	0.566638
17	1107	0.691667	1373	0.831356
18	1239	0.762580	11808	6.481448
19	1731	1.112580	5547	3.036656