Sudoku

Wadood Alam*, Matthew Pacey*, Joe Nguyen* (Equal contribution)

May 14, 2022

1 Introduction

Our goal for this paper is to solve the Sudoku Puzzle. The Sudoku Puzzle is a nine by nine grid of cells. At the start of the puzzle some cells have a value from 1 to 9 and some cells are blank. To correctly solve the puzzle, the player must fill in the missing cells with numbers 1 through 9 without repeating the same number in any row, column or the 3x3 region of 9 cells it occupies. We are using a backtracking algorithm with constraint propagation to solve each puzzle. The constraint propagation is achieved using established Sudoku rules for eliminating possible values from unsolved cells. These techniques are used to solve puzzles at four levels of difficulty: easy, medium, hard, and evil. During the solving of the puzzles, we record how often each Sudoku rule is applied.

2 Experimental Setup

Our solution is implemented using Python. The backtracking algorithm is implemented with and without inference rules for the two heuristic functions (fixed baseline and most constrained variable).Then we collect the corresponding data, displayed in section 3.

2.1 Heuristic Functions

Fixed Baseline function uses a fixed pattern of traversing. The order it follows is row-wise from top to bottom, columns from left to right, and regions top-left to bottom-right. The other function we used is called the **Most Constrained Variable** which picks the cell which has the least number of values in its domain.

2.2 Pseudocode for Backtracking Algorithm

Algorithm 1 Backtracking Algorithm

```
Backtrack(sudoku, Rules)
Inference rules applied
Get list of (heuristic value, (i, j)) // Only for MCV
Gets all the possible values for (i, j) from the puzzle
For value in possible values
Pick the value
if Cell value is valid then
    Assign the value to the corresponding cell
    Update domain values for all other cells in row, col, region
    Recursive Backtrack(UpdatedSudoku, Rules)
    Returns solved sudoku puzzle
end if
End For
Return
```

2.3 Inference Rules

We integrated the following inference rules into our code as techniques to solve the puzzles. The rules are ordered from the simplest to the most complex. When all rules are enabled, they are checked starting with the simplest and only moving to the next rule when the current rule no longer finds any matches.

Naked Singles: Assign a value to the cell if it is the only possible value for a given cell.

Hidden Singles: Assign a value to a cell which is not contained in the domain of possible values in any other cells.

Naked Pairs: An identical pair that occurs in two cells of a row, column, or region. Remove these numbers from rows, columns or boxes that share these cells.

Hidden Pairs: A pair of numbers that occurs only in two cells in a row, column, or region. Eliminate the other numbers from them.

Naked Triples: Similar to naked pairs but with three values. Of the three values, two may only appear in one of the triple cells and the naked triple is still valid. Other cells in the group can safely have all of the tripled values removed from them as possible values.

Hidden Triples: Similar to hidden pairs but with three values. In the case of hidden triples, all non tripled values can be removed from the three cells containing the triples.

3 Results

The results are summarised as follows:

Experiment Method	Easy	Medium	Hard	Evil
Fixed Baseline				
- *Number of problems Solved*	100 %	100 %	100 %	100 %
- *Average Number of Backtracks*	109	999	1741	1855
- *Average Time Taken(sec)*	0.1078	0.8830	1.6549	1.9192
Most Constrained Variable				
- *Number of problems Solved*	100 %	100 %	100 %	100 %
- *Average Number of Backtracks*	4	84	82	132
- *Average Time Taken(sec)*	0.0936	0.4995	1.2097	1.0124

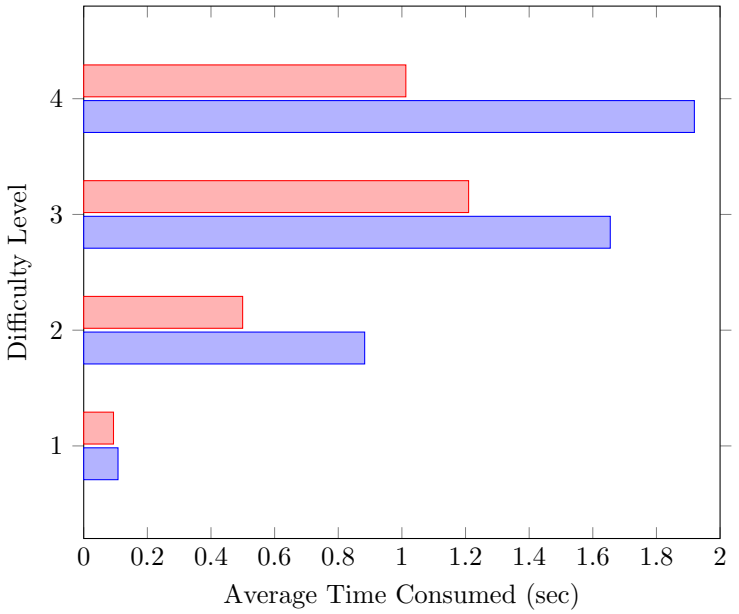
Table 1: Backtracking without Inference rules

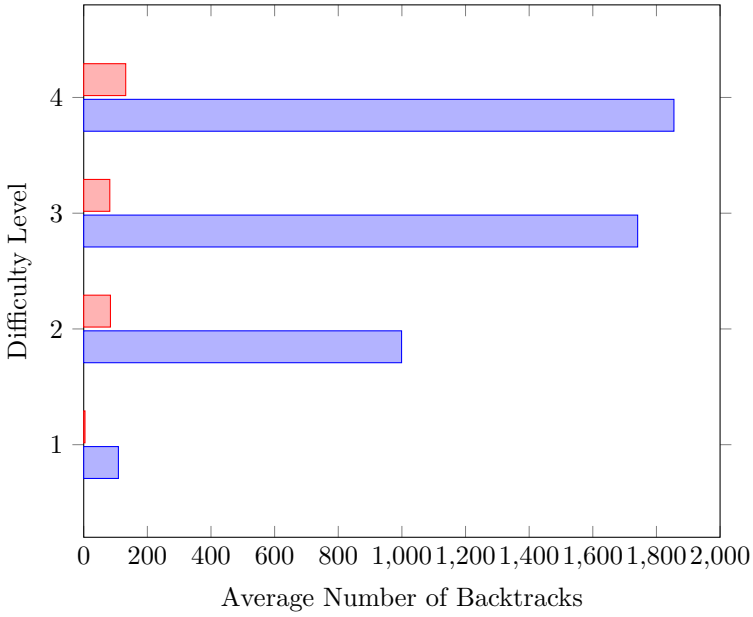
Experiment Method	Easy	Medium	Hard	Evil
Fixed Baseline				
- *Number of problems Solved*	100 %	100 %	100 %	100 %
- *Naked and Hidden Singles*	100 %	100 %	100 %	100 %
- *Naked and Hidden Singles + Doubles*	100 %	100 %	100 %	100 %
- *Naked and Hidden Singles + Doubles + Triples*	100 %	100 %	100 %	100 %
- *Average Number of Backtracks*	1	1	2	2
- *Total Singles (Naked + Hidden)*	622 + 43	238 + 552	254 + 408	340 + 243
- *Total Doubles (Naked + Hidden)*	0 + 0	8 + 0	23 + 5	27 + 8
- *Total Triples (Naked + Hidden)*	0 + 0	1 + 0	3 + 0	7 + 0
- *Average Time Taken(sec)*	0.1038	0.1193	0.1599	0.1762
Most Constrained Variable				
- *Number of problems Solved*	100 %	100 %	100 %	100 %
- *Average Number of Backtracks*	1	1	2	2
- *Naked and Hidden Singles*	622 + 43	229 + 552	267 + 379	351 + 224
- *Naked and Hidden Singles + Doubles*	0 + 0	8 + 0	26 + 5	32 + 9
- *Naked and Hidden Singles + Doubles + Triples*	0 + 0	1 + 0	4 + 0	6 + 0
- *Average Time Taken(sec)*	0.1072	0.1217	0.1610	0.1760
Forward Checking (No backtracking)				
- *Number of problems Solved*	100 %	100 %	82 %	80 %
- *Naked and Hidden Singles*	622 + 43	238 + 552	250 + 336	335 + 156
- *Naked and Hidden Singles + Doubles*	0 + 0	8 + 0	23 + 5	30 + 9
- *Naked and Hidden Singles + Doubles + Triples*	0 + 0	1 + 0	3 + 0	7 + 0
- *Average Time Taken(sec)*	0.1217	0.1374	9.0601	10.5118

Table 2: Backtracking with Inference rules

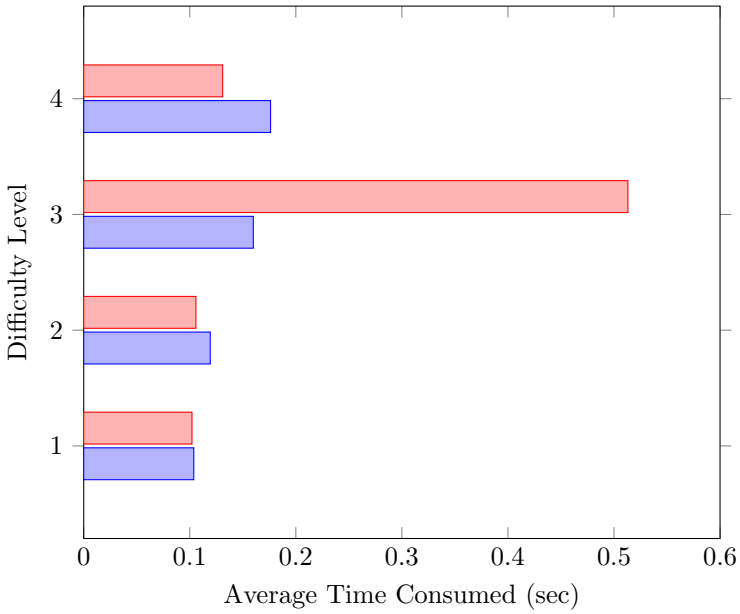
Note: Blue = Fixed Baseline , Red = Most Constrained Variable; Difficulty level: 1 = Easy,2 = Medium,3 = Hard,4 = Evil

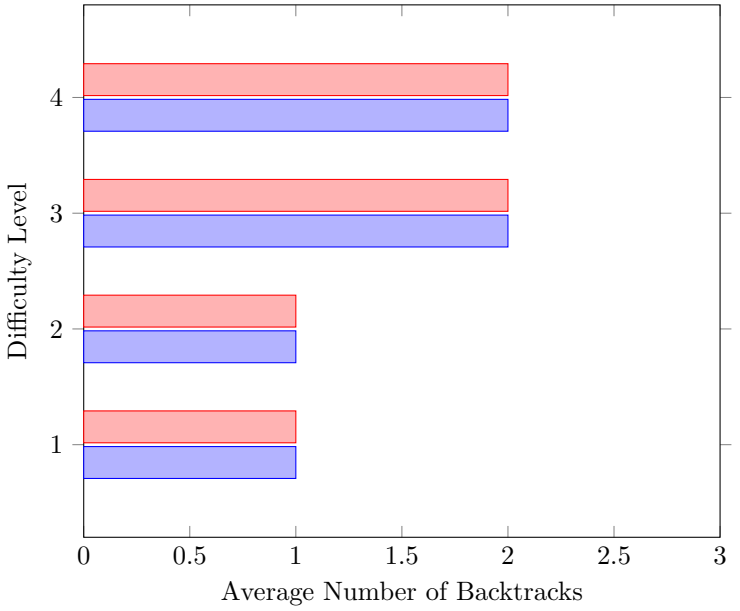
3.1 Graphs for Backtrack without inference rules





3.2 Graphs for Backtrack with inference rules





4 Discussion

4.1 Difficulty

We can grade the problems by the complexity of rules needed to solve them without backtracking, given the results of *Forward Checking* in Table 2. The number of problems solved and the level of rule's complexity are correlated to the level of difficulty. In other words, the harder the problem, the more complex rules we have to use and the fewer number of problem solved.
As the difficulty of the problem increases, the time taken to compute and number of backtracks increases as well. With increasing difficulty, the number of hiddens and nakeds that are utilized increases. The harder the problem is, the more complexity of inference rule is required. According to this data, we can grade the problems as: easy, medium, hard, and evil based by the number of triples required to solve.

4.2 Fixed baseline (FB) vs Most Constrained Variable (MCV)

4.2.1 Without inference rule

The Most Constrained Variable heuristic significantly decreases the number of backtracks and time required for computation as compared to fixed baseline. Similarly, the time required to solve the puzzles is also decreased by using a better heuristic (MCV).

4.2.2 With inference rule

1) The inference rules are strong enough to help reduce the number of backtracks in two algorithms and make these roughly the same. 2) The number of inference rules called in both are also comparable too. Given 1) and 2), the running time of MCV is comparable to FB.

4.3 Inference rules

The efficacy of the rule subsets substantially reduced the number of backtracks. Similarly, the time required to solve the puzzles is also decreased by using a better heuristic (MCV) and subset of inference rules.
The inference rules significantly reduced the time required to solve the puzzles. By using inference, the average time to solve complex puzzles was reduced to about a tenth of the time (comparing fixed baseline for evil, 1.9 seconds with no inference and 0.17 seconds with inference).