A Sudoku Solver Algorithm

Long Nguyen
Department of Electrical and Microelectronic Engineering
Rochester Institute of Technology
Rochester, NY 14623
lhn3149@rit.edu

Duc Le
Department of Electrical and Microelectronic Engineering
Rochester Institute of Technology
Rochester, NY 14623
dhl3772@rit.edu

Chongyu Zhu
Department of Electrical and Microelectronic Engineering
Rochester Institute of Technology
Rochester, NY 14623
cz7908@rit.edu

***Abstract*—The report compares three different algorithms to solve a classic 9x9 Sudoku problem. Their execution times, recursion depths and spatial asymmetries are evaluated and compared for 2 datasets containing millions of Sudoku puzzles with varying difficulties.**

Keywords—sudoku, backtracking, optimization

I. Introduction

Sudoku is a numerical puzzle that was popularized in Japan in 1984. The puzzle is a classic board game which is very popular in magazines and quizzes. Sudoku is a constraint – satisfaction problem, which involves filling numbers from 0-9 into a 9x9 block of number array so that there is no repeated number in a row, a column and a 3x3 sub square. Because of the constraint – satisfaction nature, the design thought process, techniques, and method of solving Sudoku is fundamentally important and could be applied to other real-world problems like the shortest path, optimization, and searching. [1]

The game has been studied extensively both in mathematics and Sudoku's solver algorithms. In this report, the fundamental mathematics of Sudoku is reviewed. In addition, three popular Sudoku solvers will be presented and compared. A dataset of millions of Sudoku puzzles with different difficulties is used to evaluate the efficiency of the algorithm. More importantly, we will evaluate the spatial symmetry of the algorithms by comparing the performance on the original dataset and the transposed dataset. Three algorithms yield distinctively different results on these tests.

Finally, further work towards the final report is listed together with the detailed deadline and work assignments for each teammate.

II. Background And Benchmarks

A. The Sudoku Game

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

Fig. 1: 9x9 Sudoku game

The classic puzzle starts off with a 9x9 grid with some given clues. The goal of the game is to fill numbers so that a number only repeats once across its row, column and sub squares. The clues are given so that there is only a unique solution. A paper by Gary McGuire, Bastian Tugemann, and Gilles Civario proved by an exhaustive computer search that the minimum number of clues in a proper classics Sudoku is 17 [2].

The classic 9x9 Sudoku is considered a rank 3 problem. In more generalized case, a rank n Sudoku is an n^2 x n^2 square grid, subdivided into n^2 blocks, each of size n x n. The numbers used to fill is from 1 to n^2.

There are more variations to the game such as region shapes variations, mini sudoku, alphabetical sudoku, windoku, etc.

B. *Mathematical Model*

A Sudoku grid has 6.67×10^{21} possible configurations which can be reduced to 5,472,730,538 different groups under validity preserving transformations. These transformations include rotation (4), mirroring (2), cyphering (replacing a number by another number at every position), and row and column exchange. [3]

Other Sudoku size combinations are summarized below. [1]

Grid	Blocks	Exact	Magnitude
4×4	2×2	288	2.8800×10^{2}
6×6	2×3	28,200,960	2.8201×10^{7}
8×8	2×4	29,136,487,207,403,520	2.9136×10^{16}
9×9	3×3	6,670,903,752,021,072,936,960	6.6709×10^{21}
10×10	2×5	1,903,816,047,972,624,930,994,913,280,000	1.9038×10^{30}

Fig. 2: Sukodu possible combination by size

C. *Algorithm Techniques Analysis*

There are various techniques to solve Sudoku puzzles. Without loss of generalization, we choose to solve 9x9 classics Sudoku. The general technique to approach a constraint-satisfaction problem is the AC-3 (Arc Consistency #3). The algorithm will loop through all possible values of an entry and check if the value fails any constraints. If it does not, it will continue to check the next entry.

In the Sudoku game, a block can have 9 values from 1 to 9 and 3 constraints to be satisfied: no repetition across rows, columns, and sub squares.

The simplest algorithm is the naïve approach (Algorithm 1), which scans the grid cell by cell. At a blank cell, it tries a possible value and then calls the function again for the next blank cell. If a cell has no possible values, it unfills the previous cell and tries the next candidate. A solution is reached when the last blank cell of the grid is filled.

Another technique is backtracking. It will find all possible candidates satisfying 3 constraints at a cell. Then it substitutes a value to a blank cell and calls the solving function again recursively, updating values for other blank cells. If it is unsolvable at a point, it will go back and substitute a different value, then call the solving function recursively. This recursive algorithm is considered the standard method for Sudoku solvers. In this paper, it is implemented in Algorithm 2.

Multiple improved techniques revolving around backtracking utilizes the geometry relation between blocks to reduce the number of candidates and thus the complexity. For example, a technique called Naked Single Value (Algorithm 3) will find and immediately update the value for cells with only one possible value (candidate). This technique proves to significantly reduce the execution time by reducing the number of candidates in other cells with certainty. After filling all naked single cells, we will be left with a blank cell with multiple possible values, which we will apply backtracking techniques above with fewer numbers of candidates to solve.

A method which can be used after Naked Single Value is Naked Pair/Triplet. If there are 2 repeated pairs within a group (row, column, and corresponding sub square), we can remove the values of the pair in other blank cells, effectively reducing the number of cases we need to iterate. Similarly, if there are 3 repeated triplets within a group, the values cannot be elsewhere.

In this project, 3 different techniques (naive, backtracking, backtracking with naked single value) are implemented to solve several sudoku puzzles and analyzed further.

III. Proposed Method

A. *Rationality*

The focus of the project is to optimize the Sudoku solver with respect to execution time while ensuring the accuracy. The problem arises where difficult Sudoku puzzles take a significant amount of time to solve, which makes the brute-force technique impractical. To characterize the execution time, we have a variable called level, meaning the depth of the algorithm.

B. *Pseudocodes*

1. Naïve approach (Algorithm 1):

```
function sudoku_simple(X,i,j):  %      return
true                                    %  or
false
   if reached last cell
      solution = X

   if reached end of row
      goes to next row, first column

   if current cell is filled
      return sudoku_simple(X,i,j+1)

   for candidate=1:1:9
      if candidate is safe
          X(i,j)=candidate
          if sudoku_simple(X,i,j+1)
             return true
      X(i,j)=0 % unfill cell
   return false % no candidates work
```

2. Random backtracking (Algorithm 2):

```
function backtrack_random(X):
   C = candidates(X)
   [ie,je] = indices of X with no candidates
   if [ie,je] is not empty
      return X % found solution


   if there are still unfilled cells in X
      [i,j] = indices of 1st unfilled cell
     for y = nonzero entries of C(i,j,:)
       X(i,j) = y
       X = backtrack_random(X)

function candidates(X)
   C = zeros 9x9x9 matrix
   for i=1:1:9
      for j = 1:1:9
        if X(i,j) is unfilled
           C(i,j,:) = all safe entries at X(i,j)
```

3. Backtracking with Naked Single Value (Algorithm 3):

```
function backtrack_single(X):
   C = candidates(X)
   [ie,je] = indices of X with no candidates
   [is,js] = indices of X with only 1 candidate

   while [ie,je] is empty and [is,js] is not
empty
      X(is,js)=C(is,js,1)
      C = candidates(X)
      [ie,je] = indices of X with no candidates
      [is,js]  =  indices  of  X  with  only  1
candidate

   if [ie,je] is not empty
      return X % found solution

   if there are still unfilled cells in X
      [i,j] = indices of 1st unfilled cell
     for y = nonzero entries of C(i,j,:)
       X(i,j) = y
       X = backtrack_single(X)

function candidates(X)
   C = zeros 9x9x9 matrix
   for i=1:1:9
      for j = 1:1:9
        if X(i,j) is unfilled
           C(i,j,:) = all safe entries at X(i,j)
```

IV. EXPERIMENTS

A. *Experiment Setup*

In this part, we will describe the actual input and output of MATLAB code. This will help the understanding and usability of the code.

Input to each algorithm:

- A string of 81 characters, representing the problem. Empty cells are indicated as 0.

Output of each algorithm:

- If solvable, return solved puzzle.
- If unsolvable, print out unsolvable Sudoku.
- Execution time.

Apart from the algorithm files, we have a test file for running multiple Sudoku problems data to evaluate the performance of each algorithm.

Input to the test file:

- Sudoku dataset of 1 million problems

Output of the test file:

- Histogram of execution time for 3 algorithms (1000 samples)
- Histogram of recursion depth level (for backtracking algorithms)
- Plots of number of clues vs execution time for algorithms (1000 samples)
- Histogram of time difference between solving the original problem and the transposed problem (to demonstrate spatial asymmetry)

B. *Performance measure*

Due to the solvability and simple nature of Sudoku, the algorithms will be able to achieve 100% accuracy on solving the dataset. Therefore, execution time will be our main performance measure for comparison.

C. *Baseline performance with easy dataset*

This experiment looks at the performance of the 3 proposed algorithms on a dataset of easy Sudoku problems taken from Kaggle [4] to provide a preliminary baseline on the differences between the algorithms. For 1000 problems in the dataset, all 3 algorithms provide

matching solutions, meaning that implementations of the algorithms work.

Next, we plot the distribution of the processing time of the 3 algorithms below.

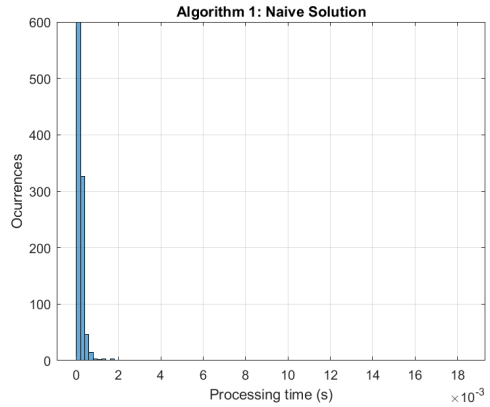

Fig. 3: Naïve execution time

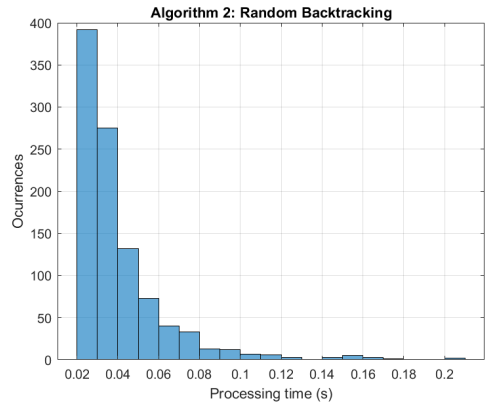

Fig. 4: Random backtracking time

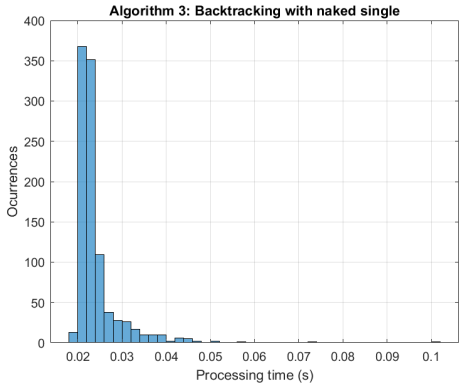

Fig. 5: Backtracking with naked single execution time

From the experiment, the naive solution takes the least amount of time. The backtracking with naked single also takes significantly less time than random backtracking, meaning prioritizing cells with single candidates has a huge impact on the backtracking algorithm.

D. Timing performance with more dataset

We then move on to a more difficult case [5] to compare the timing performance between the Naive Solution and Backtracking with naked single. Due to its high time complexity, the Random Backtracking algorithm will be omitted from this study.

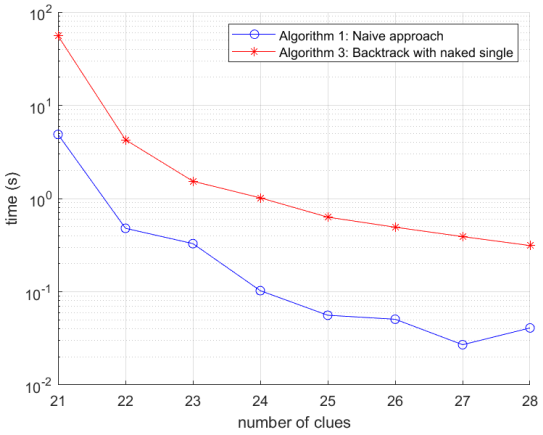

Fig. 6: Execution time vs sudoku clues

From the plot above, the time complexity of both algorithms unsurprisingly decreases with a higher number of clues, i.e., pre-filled cells. However, the naive approach unexpectedly seems to perform faster than the backtracking approach. This can be due to the inefficiency of the backtracking implementation in MATLAB, where cells are used to store candidates. Using cells is known to slow down codes in MATLAB due to its extra ability of storing different data types such as objects, strings, etc. in the same array, adding extra overhead complexity.

We also look at the maximum recursion level for the backtracking algorithm (Algorithm 3) for different numbers of clues.

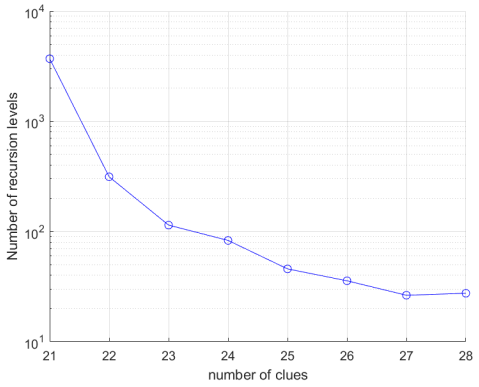

Fig. 7: Recursion level vs number of clues

From the above plot, the lower number of clues correlates to a higher number of recursions, which is expected for backtracking algorithms

E. Algorithm spatial asymmetry study

Furthermore, we assess the spatial asymmetry of the 3 proposed algorithms, which is measured by taking the difference between the processing times it takes to solve the problem and its transpose, normalized by dividing by the average of the 2 processing times. Thus, the algorithm asymmetric metric is defined by

$$A = \frac{|t - t_{transpose}|}{(t + t_{transpose})/2}.$$

The main motivation behind this study is that solving Sudoku involves scanning the problem in a certain direction, meaning the computation process is completely different under some simple spatial transformations, which realistically should not affect the computational efficiency.

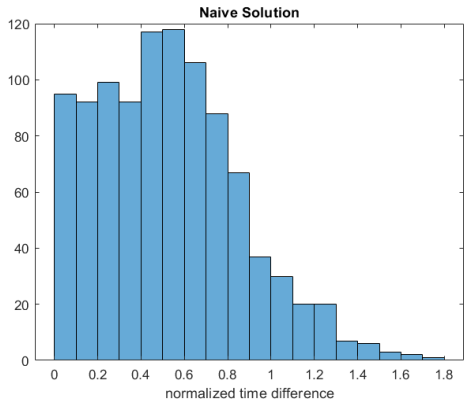

Fig. 8: Time difference histogram of naïve solution (Algorithm 1)

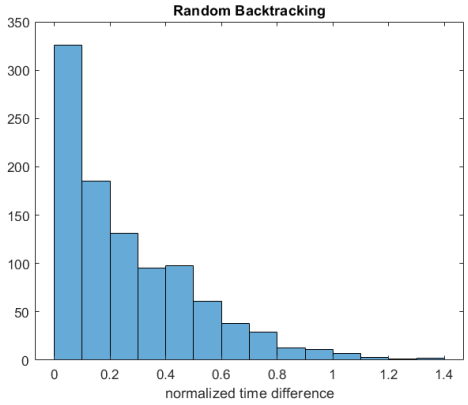

Fig. 9: Time difference histogram of random backtracking (Algorithm 2)

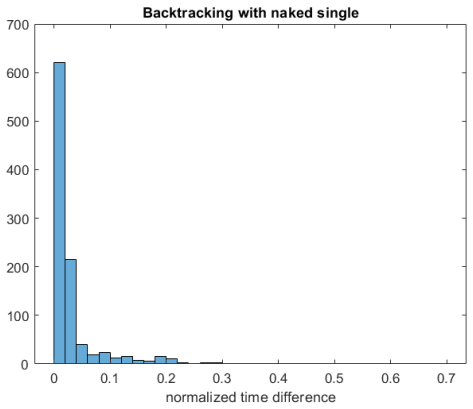

Fig. 10: Time difference histogram of backtracking with naked single (Algorithm 3)

From the above plots, the Backtracking algorithm with naked single (Algorithm 3) has the least spatial asymmetry. This can be explained by its practice of filling in the cells with a single candidate, which are distributed sparsely and more randomly across the grid. On the other hand, Naive solution and Random Backtracking involves a lot of spatial scanning, hence the high spatial asymmetry.

F. Timing performance with 16 by 16 Sudoku

We apply the algorithm to the 16 by 16 Sudoku and see the distinct difference in time performance between the algorithms. Since a large dataset of 16 by 16 puzzles is not available, we only test our algorithm for one problem.

The problem is presented below.

1	5	10	2	3	4	9	11	12	16	6	14	15	13	7	8
14	16	8	13	5	15	7	12	4	3	1	2	9	10	6	11
9	12	4	7	10	0	0	1	0	13	0	11	0	0	14	0
3	0	0	15	2	0	0	14	0	0	0	9	0	0	12	0
13	0	0	0	8	0	0	10	0	12	2	0	1	15	0	0
0	11	7	6	0	0	0	16	0	0	0	15	0	0	5	13
0	0	0	10	0	5	15	0	0	4	0	8	0	0	11	0
16	0	0	5	9	12	0	0	1	0	0	0	0	0	8	0
0	2	0	0	0	0	0	13	0	0	12	5	8	0	0	3
0	13	0	0	15	0	3	0	0	14	8	0	16	0	0	0
5	8	0	0	1	0	0	0	2	0	0	0	13	9	15	0
0	0	12	4	0	6	16	0	13	0	0	7	0	0	0	5
0	3	0	0	12	0	0	0	6	0	0	4	11	0	0	16
0	7	0	0	16	0	5	0	14	0	0	1	0	0	2	0
11	1	15	9	0	0	13	0	0	2	0	0	0	14	0	0
0	14	0	0	0	11	0	2	0	0	13	3	5	0	0	12

Fig. 11: 16 by 16 Sudoku problem

Algorithm 3 (backtracking with naked value) takes 0.426 second and brute force approach takes 28.99 seconds. This proves that algorithm 3 is much more efficient than the brute force approach for more complex problems which is expected.

V. Conclusion

In conclusion, we have implemented and examined the performance of 3 different algorithms to solve Sudoku problems using MATLAB. All 3 algorithms are capable of returning the accurate solution.

The naive algorithm (algorithm 1) has lower time complexity than the other 2 backtracking algorithms. Algorithm 3, by taking advantage of cells with a single candidate, performs better than Algorithm 2, indicating the significance of smart implementation of any constraint satisfaction problem. Furthermore, the time complexity is observed to increase with fewer numbers of clues.

In addition, we also proposed a scheme to test the spatial symmetry of the algorithms, and algorithm 3 turns out to be the most symmetric due to its practice of filling in the cells with a single candidate, which are distributed sparsely across the problem.

Acknowledgment

We would like to thank Dr. Markopoulos and TA Mayur Dhanaraj for the help on the fundamental understanding of machine learning

References

[1] *"Mathematics of Sudoku,"* Wikipedia, September, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Mathematics_of_Sudoku. [Accessed Nov. 22, 2007]

[2] G. McGuire, B. Tugemann, G. Civario, in *There is no 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration* . Cornell University, [Online document], Sep. 1st, 2013. Available: https://arxiv.org/abs/1201.0749 [Accessed Nov. 22, 2007].

[3] F. Jarvis, B. Felgenhauer, in *Enumerating possible Sudoku grids*. University of Sheffield, [Online document], Jun. 20th, 2005. Available: https://arxiv.org/abs/1201.0749 http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf [Accessed Nov. 22, 2007].

[4] K. Park, "1 million Sudoku games," Kaggle, 2016. [Online]. Available:.https://www.kaggle.com/bryanpark/sudoku [Accessed Sept. 12, 2007].

[5] D. Radcliffe, "3 million Sudoku puzzles with ratings," Kaggle, 2020. [Online]. Available:.https://www.kaggle.com/radcliffe/3-million-sudoku-puzzles-with-ratings [Accessed Nov. 22, 2021].