IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⋇ Impact Factor 7.39 ⋇ Vol. 11, Issue 7, July 2022

DOI: 10.17148/IJARCCE.2022.117110

An Improved Framework of Backtracking Algorithm to Solve Sudoku

Manjunath R[1], Balaraju G, L[2], Sumath S[3], Manjunath C R[4]

Professor, Department of CSE, R R Institute of Technology, Bengaluru, Karnataka[1]

Assistant Professor, Department of CSE, R R Institute of Technology, Bengaluru, Karnataka[2]

Assistant Professor, Department of CSE, Presidency University, Bengaluru, Karnataka[3]

Associate Professor, Department of CSE, Jain University School of Engineering and Technology (SET JU), Bengaluru, Karnataka[4]

Abstract: Sudoku is a pretty popular number game. The goal of this game is to fill a 9x9 matrix with unique numbers, and there should not be repeated numbers in each row, column, or block. There are several possible algorithms to automatically solve Sudoku boards; the most notable is the backtracking algorithm, that takes a brute-force approach to finding solutions for each board configuration. The backtracking algorithm uses an array of the legal numbers in the cell to attempt a solution before it moves on to the next cell. If a solution cannot be found, it backtracks and attempts to solve the board again with a different guess choice. The more errors the solver makes, the more backtracks it must perform, which decreases its overall efficiency and increases its effective runtime. We analysed the difference in the algorithm performance by comparing the number of recursive backtracks between sequential and randomly distributed guesses. Analysis show that using values that are given in a shuffled array significantly reduces the number of backtracks done by the solver and, as a result, improve the total effective efficiency of the algorithm as a whole.

Keywords: Sudoku solver, backtracking algorithm, algorithm design and analysis

I. INTRODUCTION

Sudoku is a pretty popular number game. The goal of this game is to fill a 9x9 matrix with unique numbers, and there should not be repeated numbers in each row, column, or block. Sudoku puzzle solving has been shown to belong to the category of NP-complete problems [1]. However, this is true for the general $n^2 \times n^2$ Sudoku boards. For the case of the popular Sudoku game of 9×9 matrix, the problem is reduced to have finite number of solutions. These boards can be solved with brute-force algorithms.

While the backtracking algorithm is relatively simple, its runtime optimization is usually said to depend on the implementation of its mechanics; that is, on the efficiency of finding new empty squares and on that of finding available numerical solutions per empty square. This paper will concentrate on a different element that affects the optimization of the backtracking algorithm in solving Sudoku games: the manner in which the algorithm chooses which of the available guess to plug into the board.

The most noticeable difference between the choices of guesses is the type of solutions that are produced. When the algorithm chooses its guesses in sequential order, the solutions given are deterministic; the same solution - with the same backtracks and recursions - will appear for the same initial board configurations. If the choice of guesses is shuffled, the solutions are varied, and so is the number of recursions and backtracks that the backtracking algorithm performs.

This paper will discuss the effect of choosing ordered versus shuffled guess choices behaviour on the number of recursions and backtracks.

II. THE MECHANICS OF SUDOKU PUZZLES

Let us understand how Sudoku works before analysing the backtracking algorithm. Sudoku is played on a grid of 9 x 9 spaces. Within the rows and columns are 9 “squares” (made up of 3 x 3 spaces). Each row, column and square (9 spaces each) need to be filled out with the numbers 1-9, without repeating any numbers within the row, column or square.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁜ Impact Factor 7.39 ⁜ Vol. 11, Issue 7, July 2022

DOI: 10.17148/IJARCCE.2022.117110

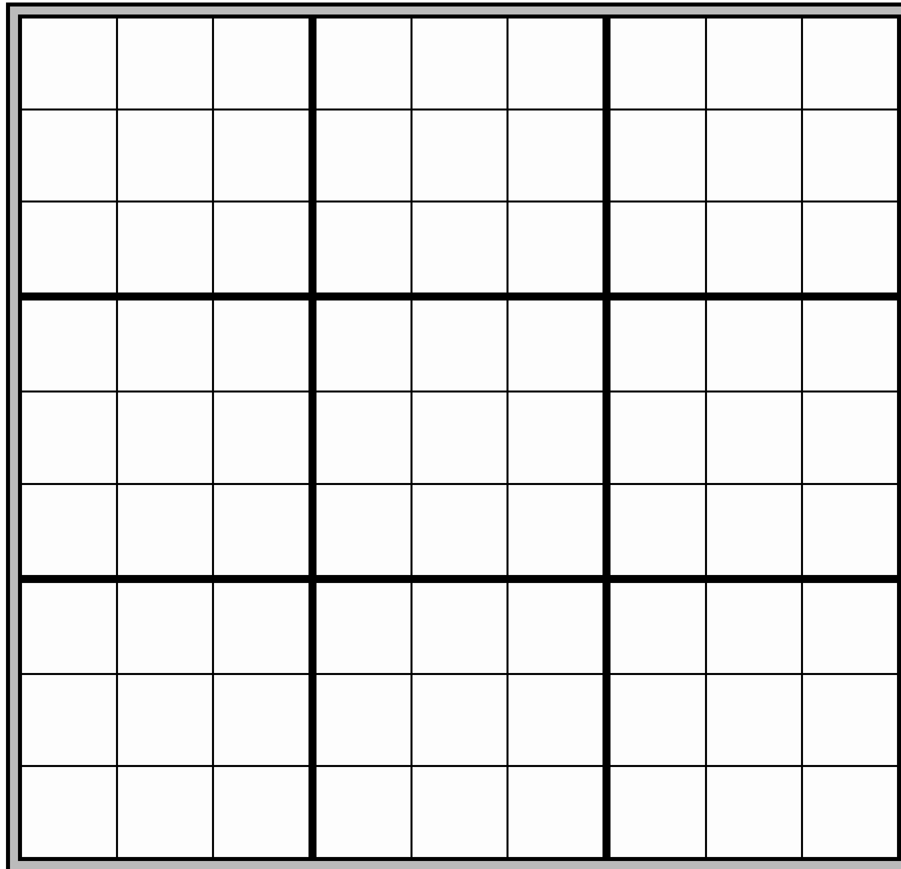

Figure 1: The classic sudoku board: a 9×9 matrix with 3×3 subsections

When we consider the possible values a certain cell can hold, we must evaluate first the cells in the same row, column and segment. If any of those cells contain a number, that number is removed from consideration as a valid choice for the current empty cell. This results in an array of legal numbers that we are allowed to consider for the current empty cell. When solved by hand, players usually mark the list of available numbers at the edge of each cell and then remove these as more and more values are filled in, resulting in smaller available numbers to choose for each cell. The game is won when all cells are filled in with legal values.

III. THE ALGORITHM

A. Implementation:

We will focus on a simple implementation for the analysis, the pseudo code is based on the explanation in [3] and [4].Algorithm:

1. Create a function that checks if the given matrix is valid sudoku or not. Keep HashMap for the row, column and boxes. If any number has a frequency greater than 1 in the HashMap return false else return true;
2. Create a recursive function solve() that takes a grid and the current row and column index.
3. Count recursion
4. Check some base cases. If the index is at the end of the matrix, i.e. i=N-1 and j=N then check if the grid is safe or not, if safe print the grid and return true else return false. The other base case is when the value of column is N, i.e j = N, then move to next row, i.e., i++ and j = 0.
5. if the current index is not assigned then fill the element from 1 to 9 and recur for all 9 cases with the index of next element, i.e., i, j+1. if the recursive call returns true then break the loop and return true.
6. if the current index is assigned then call the recursive function with index of next element, i.e., i, j+1

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⋇ Impact Factor 7.39 ⋇ Vol. 11, Issue 7, July 2022

DOI: 10.17148/IJARCCE.2022.117110

IV. CASE STUDY

This case study took references from the paper of Moriel Schottlender [2]. Two groups of tests were performed to gather information about the effect of the two different types of guess choices. The first test was built around the deterministic solution for an empty board as seen in figure 1. The second test involved gathering information about the amount of backtracks in multiple non deterministic solutions as bases. The second test was then repeated for the case of a guess choice array that is ordered in reverse; the results were the same for both ordered arrays, so only one is represented in the results and analysis.

The counting of recursions and backtracks were done in the algorithm itself, and are shown in the pseudo- code in the algorithm. Recursions are counted on every iteration of the solve() method, which also includes abacktrack; the pure recursion count would be the recursion count that results from the operation minus the backtracks. However, in the best-case scenario for an empty board, there will be as many recursions as there are empty cells, and any additional recursions will occur only because of backtracks. Each backtrack will result in at least one recursion. The two counters are related, and the performance is directly affected by the number of backtracks in each case.

Both guess choices were tested with boards that varied in the number of prefilled numbers. The boards started empty, then were tested against boards with 5, 10, 15, 20 prefilled numbers taken from the base solution and filled into sequential empty cells on the board. Each configuration was then solved the data about recursions and backtracks saved for analysis.

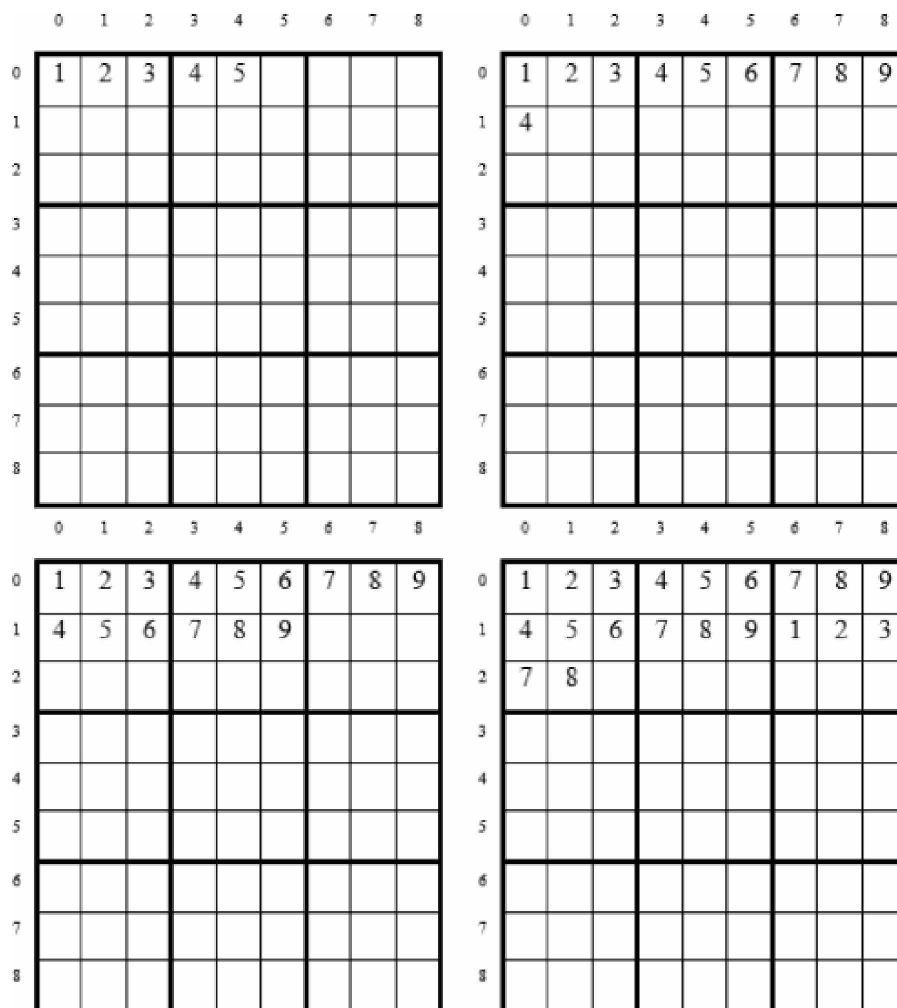

Figure 2: Initial board configuration using the deterministic solution as a base, with 5, 10, 15 and 20prefilled numbers [2]

 This work is licensed under a Creative Commons Attribution 4.0 International License

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified ⁘ Impact Factor 7.39 ⁘ Vol. 11, Issue 7, July 2022
DOI: 10.17148/IJARCCE.2022.117110

For consistency, the initial conditions included prefilled numbers from the base solution, filling the board sequentially from the top-left of the board. Each board was then solved first with ordered and then with shuffled guess choices.

When ordered guess choices were used, only one solution was obtained to collect the number of recursions and backtracks performed; there was no need to test more than once, since that method in that case is deterministic, and produces the same result for every initial case.

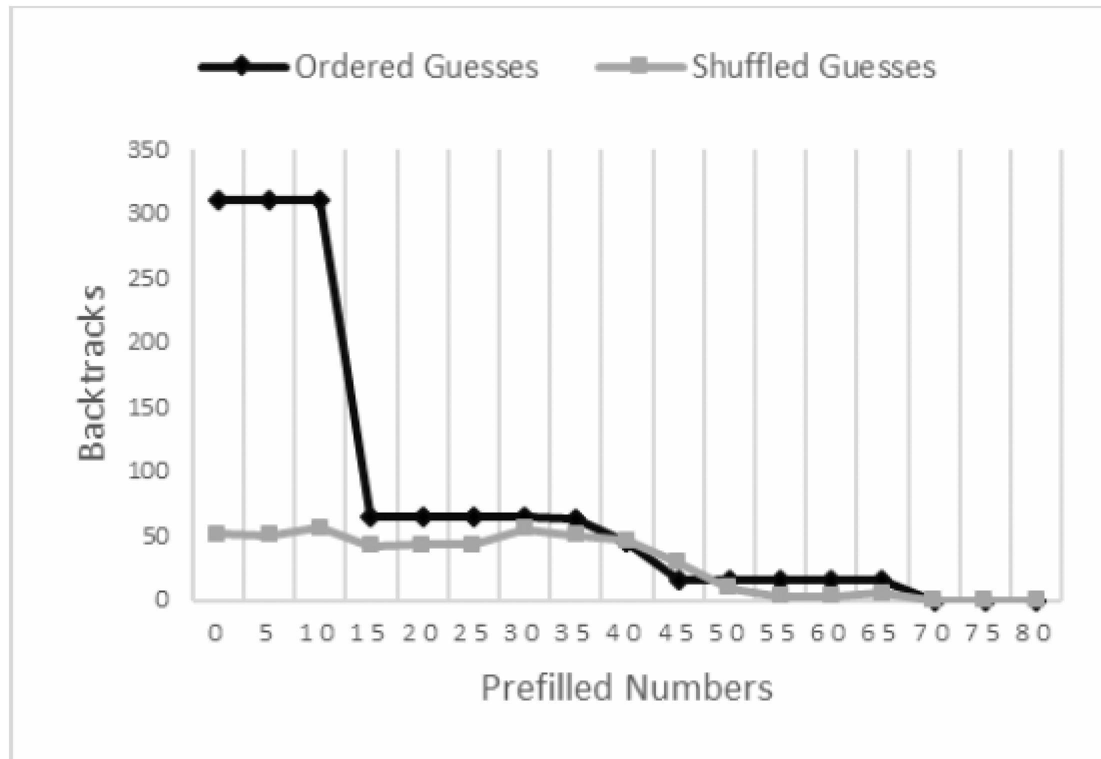

Figure 3: Amount of backtracks per prefilled values [2]

For the case of shuffled guesses, each board configuration was solved 10,000 times, and the average recursions and backtracks was obtained for each configuration. The number of attempts was purposefully large to make sure the standard error is in acceptable levels.

The first test results showed a clear difference in performance as represented by the amount of backtracks required between ordered and shuffled guess choices. Figure 3 shows the results of the number of backtracksrequired to solve a board with different configuration of prefilled numbers for ordered and shuffled guesses. The difference is especially notable for boards with up to 15 prefilled numbers, and becomes roughly equal in both methods for boards with 30 prefilled numbers and above.

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁜ Impact Factor 7.39 ⁜ Vol. 11, Issue 7, July 2022

DOI: 10.17148/IJARCCE.2022.117110

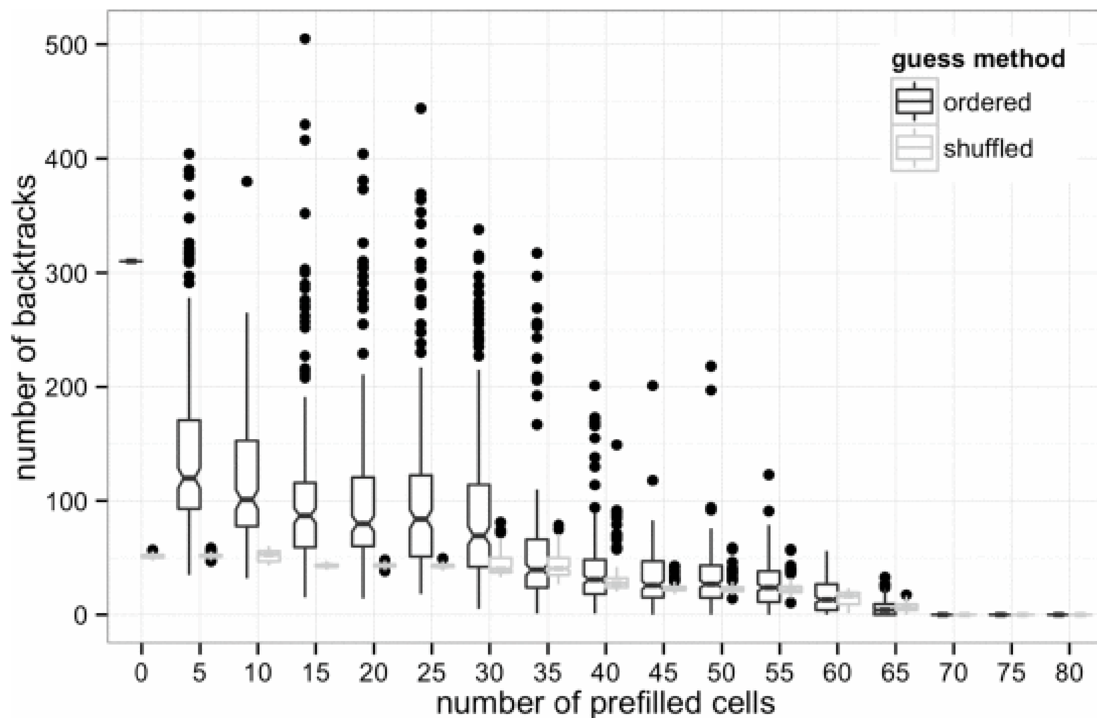

Figure 4: The distribution of backtracks across all 195 solutions and prefilled number configurations for bothordered and shuffled methods [2]

This showed a sufficiently significant difference to perform further tests with more solutions as bases. Figure 4 shows the average number of backtracks over the 195 solutions for shuffled and ordered guesses, and demonstrates the same general effect; the algorithm requires significantly fewer backtracks on average for up to 35 prefilled numbers, after which the two methods are roughly equal.

V. ANALYSIS

After examining, the shuffled approach has a bigger chance of accidentally picking the most optimized solution for the entire board in each given cell. It may theoretically, by chance alone, solve certain percentage of the board in a manner that requires no corrections. Overall, then, using the shuffled approach has a much better potential of producing solutions with fewer backtracks.

This is proved by examining the number of backtrackings required for ordered vs shuffled choices. It's clear that performance of the algorithm is much better in shuffled solutions with their range of backtracks smaller and on a lower range than that of ordered guesses.

VI. CONCLUSION

While optimization of the algorithm was not discussed, there is a relevance influence to the choice of ordered versus random guesses. Using the shuffled guess array produces a significantly better performance for the algorithm than using an ordered array for guess choices; this effect should be taken into considerationwhen discussing optimization to the Sudoku backtracking algorithm.

It should be noted that this paper is not the first to consider the effect of randomness on solving Sudoku puzzles. The explanation offered in [5] can also help explain the results in this paper, namely that using randomness as a factor, difficult puzzles can be solved as efficiently as easy ones, because the methodology works by going over random choices - either choice of empty cells, or guesses, or general approach to the brute-force algorithm.

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified ⁒ Impact Factor 7.39 ⁒ Vol. 11, Issue 7, July 2022

DOI: 10.17148/IJARCCE.2022.117110

REFERENCES

1. T. Yato, T. Seta, "Complexity and completeness of finding another solution and its application to puzzles," IEICE Trans. Fundamentals, Vol E86-A, No 5, p 1052-1060. 2008 http://www-imai.is.s.utokyo. ac.jp/~yato/data2/ SIGAL87-2.pdf
2. M. Schottlender, "The effect of guess choices on the efficiency of a backtracking algorithm in a Sudoku solver," IEEE Long Island Systems, Applications and Technology (LISAT) Conference 2014, 2014, pp. 1-6, doi: 10.1109/LISAT.2014.6845190.
3. M. Schottlender. "Designing a Javascript Sudoku puzzle: an adventure in algorithms." SmarterThatnThat.com. 1 Feb 2014.Web. 5 Mar 2014.
4. GeeksForGeeks. "Sudoku | Backtracking-7" https://www.geeksforgeeks.org/sudoku-backtracking-7. 17 Jun, 2022
5. M. Perez and T. Marwala, "Stochastic optimization approaches for solving Sudoku," School of Electrical & Information Engineering University of the Witwatersrand. 13 pages; 6 May 2008. doi:10.1016/j.eswa.2012. 04.019.

© IJARCCE This work is licensed under a Creative Commons Attribution 4.0 International License