Sudoku Solving using Depth-First Search and various Constraint Optimization Techniques

Marvin Üürike
mu07738@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

Martin Rode
mr37804@student.uni-lj.si
Faculty of Computer and Information Science,
University of Ljubljana
Večna pot 113
SI-1000 Ljubljana, Slovenia

ABSTRACT

This paper presents the development and evaluation of an AI Sudoku solver employing the Depth-First Search (DFS) algorithm augmented with various constraint optimization techniques. Our goal was to improve the solver's efficiency and accuracy. The solver was enhanced by introducing methods such as pre-computing and updating legal values for cells, prioritizing cells with the fewest candidate values, and incorporating strategies like hidden singles and naked pairs. We also experimented with prioritizing value search based on occurrence frequency.

KEYWORDS

Sudoku, Depth-first search, Constraint optimization

ACM Reference Format:
Marvin Üürike and Martin Rode. 2024. Sudoku Solving using Depth-First Search and various Constraint Optimization Techniques. In *Proceedings of Student Computing Research Symposium (SCORES'22)*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Sudoku is a popular number puzzle game that originated in Japan. The objective of the game is to fill a 9x9 grid with digits from 1 to 9 in such a way that each column, each row, and each of the nine 3x3 subgrids (also known as "boxes") contains all the digits from 1 to 9 without repetition. The puzzle starts with some cells pre-filled with numbers, and the player must fill in the remaining cells following these rules.

In this project, we aim to develop an AI Sudoku solver using the Depth-First Search (DFS) algorithm combined with various logical constraints. The goal is to enhance the solver's efficiency and accuracy in providing solutions to any valid Sudoku puzzle and to compare the effectiveness of different logical constraints in speeding up the solving process.

SCORES'22, October 6, 2022, Ljubljana, Slovenia
© 2024 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 THE ALGORITHM

This problem is a classic Depth-First Search problem. In the basic implementation, the solver inserts numbers from 1 to 9 into cells row by row, trying each possibility until a solution is found or there aren't any legal options explore - then it backtracks. This method is simple but may lead to inefficient exploration of the search space, thats why we tried to iteratively improve the algorithm with a couple of techniques.

2.1 Original Legal Values

First we introduced a way to keep track of the legal values for each cell. Before solving the puzzle, the solver calculates the legal values for each cell based on the initial puzzle configuration and stores them in a cache. During the solving process, the solver only considers these pre-computed legal values for each cell. Even without updating during solving, this reduces the search space compared to the basic approach.

2.2 Updating Legal Values with Cell Prioritization

The next logical step was to update the cache every time a value was inserted into the sudoku. Every time a number is inserted it makes that number in every cell that shares a row, column or block with it, so we must remove them from the cache. To fully take advantage of the cache we changed how the solver chooses which cell to fill next - instead of going row-by-row it chooses the cell with the least candidate legal values. This proved to significantly improve the solving times.

2.3 Hidden singles

After inserting a value into a cell, the solver updates the cache of legal values for other cells directly affected by this entry (values that aren't legal anymore). But inserting a value can also have an indirect effect. We can check each row, column, block to see if we created any so called *hidden singles*. These are candidate values that only have one legal spot left in a row, column or block. If so, we can be sure that it belongs there even if that cell still has other values currently deemed as *legal*. The excessive legal options for that cell can be removed from the cache, accelerating subsequent iterations. Hidden singles represent efficient opportunities for progression in solving the puzzle, providing definitive solutions without the need for further exploration or backtracking.

2.4 Naked pairs

Naked pairs are a pattern that can emerge when solving - similarly to *hidden singles*. As the name suggests it involves pairs of numbers. If two cells in the same row, column or block have the exact same pair of legal values (without any others), we can be sure that these two values must be in these cells. Knowing that we can remove them from all the other cells in the same unit, again reducing the search space.

2.5 Priority value search

At the end we tried to add on last approach that doesn't involve shrinking the search space through the cache. Once a cell is selected, this strategy prioritizes trying the legal values of a cell based on their occurrence frequency in the legal values of other cells within the same row, column, or block. Starting with the least occurring values should increase the likelihood of choosing the correct one. By prioritizing values with lower occurrence frequencies, the solver strategically explores fewer common possibilities first, potentially leading to faster solutions.

3 RESULTS

To test the implementation of strategies 300 *hard* sudokus were downloaded online and solved. For each algorithm and each sudoku the number of calls to the recursive solver and total time were computed.

It should be noted that in the current scenario, the number of function calls was restricted to 100,000 for the basic strategy and 500,000 for the others. This is due to the fact that the basic solver with the looser restriction took too long to compute. It is observed that with the Basic and Original Legal Values strategies, multiple Sudoku puzzles remained unsolved within these limits. Consequently, this limitation could skew the average number of calls downward compared to scenarios where no call limit was imposed. However, the introduction of smarter strategies resulted in a significant reduction in the number of calls required to solve 100% of the Sudoku puzzles. As intended each iteration lowered the number of calls needed to find the solution. Interestingly the priority value search didn't lower the average number of calls - it did however lower the number of calls for the hardest puzzle. This means it probably isn't worth the extra computing, except mybe for really hard puzzles.

Looking at the running times, a significant improvement is evident when comparing the simple Original Legal Values strategy to the Basic strategy. This is expected, as the former involves far fewer checks to verify if value criteria are met. Another tenfold improvement is observed with the Updating the Cache of Legal Values strategy. From there on the improvements are more marginal as the the additional processing required to identify hidden singles or count the frequencies of possibilities slows down the algorithm - barely outweighing the smaller search space. This can be also seen in the minimal times which increase in the more complex algorithms. The time gain is won in the hardest puzzles. As with the number of calls the Priority value search does not gain time in the average - but does bring down maximal time.

4 CONCLUSION

In summary, our exploration of various Sudoku solving strategies has shown significant improvements in solving efficiency. Simple approaches like Original Legal Values reduced solving time substantially, while more advanced techniques like Updating the Cache of Legal Values achieved even greater efficiency gains. Although some strategies incurred slightly longer solving times due to additional processing, their effectiveness in more complex problems underscores the importance of exploring diverse solving approaches.

Table 1: Call data.

Technique	Avg calls	Min calls	Max calls	Solved	Failed	Call limit
Basic	10669	54	99660	231	69	100000
Original Legal Values	43705	52	490894	250	50	500000
Updating Legal Values	3179	48	156323	300	0	500000
Hidden Singles	689	48	33507	300	0	500000
Naked Pairs	216	48	13242	300	0	500000
Priority Value Search	230	48	11493	300	0	500000

Table 2: Solving time data

Technique	Avg time [s]	Min time [s]	Max time [s]	Solved	Failed	Call limit
Basic	2.2669	0.0020	35.7436	231	69	100000
Original Legal Values	0.1564	0.0004	1.9215	250	50	500000
Updating Legal Values	0.0231	0.0004	1.1165	300	0	500000
Hidden Singles	0.0215	0.0014	1.0200	300	0	500000
Naked Pairs	0.0158	0.0020	0.9969	300	0	500000
Priority Value Search	0.0179	0.0022	0.8782	300	0	500000