Sudoku by Backtracking

Analysis and Design of Algorithms

Seif Yehia Sallam ID: 900193668
Salma Ahmed Aly ID: 900203182
Sara Maged Nassef ID: 900202369
Muhammad El-Mahdi ID:900202967

The American University in Cairo

Abstract

Sudoku puzzle is one of the most interesting logic based games with various levels which give our brains a good workout and make them active. Nonetheless, it can be quite difficult to grasp the rules of the game and think of solutions to the puzzle. Like all logic based games you have to train your brain to be able to detect patterns and solve a puzzle. Hence, the aim of this paper is to give insight to sudoku players and enthusiasts about the logic and algorithm of the game in order to understand the moves they are supposed to make and why they are making them. We will be focusing on one algorithm in particular: backtracking. Backtracking is a useful algorithm for solving problems with recursion by building a solution incrementally. Backtracking is particularly helpful when solving constraint satisfaction problems such as crosswords, verbal arithmetic, and Sudoku. Furthermore, this paper analyses not only the backtracking approach but also the heuristic approach, compares them and proposes a conclusion and claims as to which is better.

Introduction

Sudoku is one of the most interesting and popular puzzle games of all time; it was invented by Howard Garns in 1979 as “Number Place”. After Garns’s death, the puzzle was introduced in Japan by Maki Kaji, president of the Nikoli puzzle company, in the paper Monthly Nikolist in April 1984 as Sūji wa dokushin ni kagiru, which can be translated as ”the digits must be single”, or as ”the digits are limited to one occurrence”. The name was later abbreviated to Sudoku, as the name started catching on and the game grew in popularity, “Sudoku” became a registered trademark in Japan and soon the game spread amongst international newspapers all over the world.

Motivation

With a myriad of algorithms solving various puzzles of varying difficulties, it is not a surprise that it inspired computer scientists and engineers to automate the solution of this puzzle. There are numerous approaches that find the final solution to this problem—this paper will highlight some of these algorithms, specifically algorithms implemented by backtracking. Being a logic puzzle makes Sudoku an excellent brain game. Some studies such as the one Broker et al. have conducted at the University of Exeter and KCL have gone as far as to show that "if you play Sudoku daily, you will soon start to see improvements in your concentration and overall brain power". It's crucial to encourage the leaders of tomorrow to indulge in activities such as Sudoku rather than mindless activities that weave no benefits to them. By demonstrating the intriguing nature of the game and its dynamic abilities we are perpetuating a culture of mental enrichement. Furthermore, creating a Sudoku algorithm is considered a challenging task, especially for new programmers, and we thought it would be fun to give it a try and learn something new in the process. This paper will facilitate the understanding of the different algorithms and their performance with regards to time and space complexity. It will also be a great start for people who want to explore the techniques with which you can solve the puzzle, and help the bright minded create their own algorithm.

Problem Defintion

Sudoku is a $n \times n$ grid, usually $n = 9$, refered to as $G[n][n]$. The goal of Sudoku is to fill a 9×9 grid with numbers so that each row, column and 3×3 section contain all of the digits between 1 and 9. The n gird is partioned into n subgrids each of size $m \times m$. The player must set exactly one number from the Set $N = \{1, \ldots, n\}$ such that each row and each column should have one distinct instance of each element of the set as well as the subgrid.

Listing the constraints of the puzzle:

- Each row and column must have numbers x such that $1 \leq x \leq n$.
- $x_{ij} \neq x_{ik}$ where $j \neq k$,$\forall i \in N$.
- $x_{lj} \neq x_{qj}$ where $l \neq q$,$\forall j \in N$.

The constraints can be explained as follow: before assigning a number, check whether it is safe to assign. Check that the same number is not present in the current row, current column and current 3X3 subgrid. After checking for safety, assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then try the next number for the current empty cell. And if none of the number (1 to 9) leads to a solution, return false and print no solution exists.

Solving any Sudoku game that has $m \geq 4$ is usually hard to be done manually, and this is where these algorithms come very handy. The aim of the algorithms researched in this paper is to solve this puzzle in n size.

Literature Review

The algorithms discussed in this paper are enumerative algorithms, where they all assign to each cell a value according to the rules of the game. If at any moment the value chosen for the cell is incorrect or infeasiable, backtracking comes in handy where it backtracks until it reaches a point where it can place a new cell. There are several methods to solve this problem, Crook's algorithm "*Pencil and Paper*" algortihm, "*Simple Backtracking*" algorithm, a variation of the Simple Backtracking called "*Backtacking with heuristic moves*", "*Metaheuristic of Lewis*", "*The three-index model*", and a more compact version of it "*The two-index model*".

Koucky(2002) defines the backtracking games, including Sudoko, saying Parity games are generalised in backtracking games. A parity game is just a backtracking game without the backtracking places. Since victorious tactics in parity games are decided by positional (i.e. memoryless) methods, the issue arises whether this remains true for backtracking games.

One academic article in particular that piqued our interest was published in the Information and Decision Sciences journal titled Proceedings of the 6th International Conference on FICTA that provided us with computer approaches based on the sudoku solver, its proposed framework, and the process to extract cells. We also used the source to gather insights out of personal interest: the auto-detecting and recognizing of digits and the propagation constraint algorithms to generate the sudoku hints and solutions utilized. This can be applied to what we learned in the course about backtracking and the differentiator between it, a brute force approach, and an exhaustive approach. The information extraction helped us with the creation of the algorithm and the analysis of sudoku as a problem and puzzle that can be solved by a simple piece of code. In addition, the section of the journal dedicated to sudoku acted as a blueprint for what to include and what we can talk about.

According to Coelho and Laporte study about a comparison of several enumerative algorithms, where they compared between the *Simple Backtracking* algorithm and the Heuristic algorithm. The Hueristic algorithm performed much better on their tests than the Simple backtracking algorithm. The performance differnece was not great, as they both have very similar results when used to solve the puzzle with differnet sizes.

On the other hand,Johan and Kristofer in their Bachelor's thesis in KTH univerisity proposed very similar results as the ones Coelho and Laporte reported in their study. The most impressive algorithm of the four was the *two-index model*. It is however not going to be studied in this paper since it is a non-linear integer programming model and doesn't follow the backtracking algorithms.

Also, Maji and Pal(2013), in their study highlight a way to solve the backtracking soduko algorithm saying that a list of probable values for each blank cell is initially compiled based on the available hints. Then, using various approaches such as naked single, hidden single, lone ranger, and locked candidate, we remove the limitless opportunities of each and every blank cell, meeting the criteria that each row, column, and minigrid contain. For the naked single method, we define a naked single as the value that exists in a vacant cell if there is just one considerable potential. We acquire the naked singles and their positions after assigning the likely values to each blank cell, so we may immediately apply these values to these cells. The digits (or naked singles) are then removed from each relevant row, column, and minigrid. As a result, after removing these values, we receive a changed (reduced) status for each blank cell. On the other hand, a hidden single is a bit different. Perhaps there are blank cells with just one conceivable value depending on the scenario, but a simple exclusion of candidates in the cell's row, column, and minigrid does not reveal this. A concealed data point is a type of potential value.

The third method, lone ranger, is a term that refers to a variety which was one of several output permutations for a single blank cell in a row, column, or minigrid. The last method, which is Locked candidate, is an approach which is used a minigrid can often be seen in which a number's sole potential place is in one row (or column) inside such a block, despite the fact that the number's position is not set. A candidate with such number is referred to as a locked candidate. We may remove that number as a likely candidate along the same row (or column) in plenty of other minigrids since the minigrid must include the number in a row (or column).

Analysis and Critique of Methodologies

Introduction

In this work, all of the algorithms created and compared are enumerative. There are two backtracking techniques in which numbers are allocated to cells in an iterative manner for as long as possible. According to the game's rules, each assignment immediately limits the alternatives accessible for some of the remaining cells. Backtracking occurs if an infeasibility is discovered. The first method evaluates the cells in a predetermined sequence and assigns the numbers in ascending order. The second method prioritises the cells with the fewest possible assignments remaining.

Algorithms' discussions

Simple Backtracking

The backtracking algorithm's initial implementation varies from a brute-force technique in that it validates the possible assignments for every empty cell. The procedure goes backwards and adjusts the allocation of an already filled cell when a cell has no feasible assignment. It evaluates cells in order and gives values to empty ones beginning with the lowest possible number in relation to all other allocations, while keeping the game's principles in mind.

Psuedocode for this algortihm:-

```
1. initialize the grid with the givens

2. create a cell C
```

3. assign the value $F(C)$ to the cell
4. initialize all the empty cells with $F(C) = 0$
5. Create a boolean function $S(C, i)$ to check if the number i can be assigned to the cell
6. select the next empty cell
7. for $i = F(C) + 1$ to n
8. $F(C) = 0$
9. if$(S(C, i))$
10. $\therefore F(C) = i$
11. Go to 15
12. if$(F(C) = 0)$
13. backtrack
14. Go to 7
15. if There are empty cells
16. Go to 6
17. The suduko is solved

Time and Space Complexity

The time complexity of this algorithm is $O(9^{N*N})$
Space complexity is $O(N^2)$

Heuristic moves in a backtracking algorithm

The second version of the backtracking method takes into account the puzzle's rules as well as any numbers that have previously been allocated to other cells. Rather than visiting the cells in order, the next cell to be visited is the one with the fewest number of remaining eligible assignments. Our aim is to limit the number of times we re-evaluate assignments.

Pseudocode:-

```
1. initialize the grid with the givens
2. create a cell C
3. assign the value $F(C)$ to the cell
4. Create a boolean function $S(C, i)$ to check if the number i can be assigned to the cell
5. initialize all the empty cells with $F(C) = 0$
6. **for** all empty cells **do**
7. select the one with the least number of candidates
8. break ties randomly
9. **for** $i = F(C) + 1$ to n **do**
10. $F(C) = 0$
11. if $S(C, i)$ then
12. assign $i$ to $c$;
13. Go to 20
14. **end if**
15. **end for**
```

```
16. if $F(c) = 0$ then
17. backtrack
18. Go to 9
19. **end if**
20. **if** there are empty cells **then**
21. Go to 6
22. return the suduko Grid
```

Time and Space Complexity

The time complexity of this algorithm is $O(9^{N*N})$
Space complexity is $O(N^2)$

Applications

Most people regard Sudoku as a game of mind. They play it for the clarity of mind and to get rid of the negative thoughts by pouring it in a quite challenging puzzle. With the spread of the game and into the hands of the average people, they have created a championship for it. People are now able to make a living out of the game, not only in playing it but also in constructing the grids. There are very few people in the world who can expertly develop the game board and therefore, they are now trying to create awareness about the game and its benefits. That is why Sudoku can serve as both a career path and a path to success in many other professions.In addition, there are applications of Sudoku in the world of computer science.

- Beginners use it as a challenge to solve the algorithm on their own and learn more about the analysis and design of algorithms. The algorithm can be implemented by means of AI where they create their own Sudoku solver. Sudoku algorithms are actively used in Artificial Intelligence to train bots for various tasks. These grids and the logic behind them helps developers train bots to understand human behavior and adapt to it.
- It is also used for text encryption and image encryption.
- It can be connected to different problems such as salesman problem, a colouring problem, which is very popular in graph theory and many more.

Conclusion

The difference in performace of the simple backtracking algorithm and the one with heuristic moves is minor, yet performs better in larger numbers of N. The overhead of the former algorithm is higher than the second one, and the hueristic moves creates less dependence on the backtracking and devides it with the dependence on heuristic moves as well.

This backtracking algorithm is an interesting to practice and solve for beginners in programming. The code is simple and helps the programmer to further more understand the backtracking techniques in programming.

Acknowledgement

We would like to thank our professor Amr Goneid for his clear explaination of the different aspects of the course and his Teacher Assistant Mr. Mohamed Hany for helping and guiding us throughout the semester.

References

A. K. Maji and R. K. Pal, ”Sudoku solver using minigrid based backtracking,” 2014 IEEE International Advance Computing Conference (IACC), 2014, pp. 36-44, doi: 10.1109/IAdCC.2014.6779291.

Coelho, L. C., & Laporte, G. (2014). A comparison of several enumerative algorithms for Sudoku. The Journal of the Operational Research Society, 65(10), 1602–1610. http://www.jstor.org/stable/24505021

Crook, J.F. A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. https://www.ams.org/notices/200904/tx090400460p.pdf.

Dawar, A., Grädel, E.,; Kreutzer, S. (2005, November 8). Backtracking games and inflationary fixed points. Theoretical Computer Science. https://www.sciencedirect.com/science/article/pii/S0304397505006766?via

Ekstrom, J., Pitkajarvi, K. (n.d.). The backtracking algorithm and different representations for solving Sudoku Puzzles. Retrieved December 8, 2021, from https://www.diva-portal.org/smash/get/diva2:721641/FULLTEXT01.pdf.

Tuan, N. T., Sang, N. T., & Luu, N. C. (2019). Learning to Solve Sudoku Problems with Computer Vision Aided Approaches. 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), 539-546. doi:10.1109/codit.2019.8820660