SudoKolor: Optimizing Sudoku with Graph Coloring

George Liu, Daniel Bao, Annika Joshi

1 Abstract

Sudoku is a logic-based game played on a 9 by 9 grid. The goal is to fill in each box with a digit between 1 and 9 so that each column, row, and 3 by 3 subgrid contains each digit exactly once. Beyond its recreational facade, Sudoku, a popular combinatorial number-placement puzzle, exhibits relationships with various real-world problems, such as logistics optimization, resource allocation, and scheduling challenges. Its mathematical and algorithmic principles find applications beyond the puzzle grid, contributing to the development of efficient solutions in practical domains. While naive approaches, such as brute force and backtracking, are most commonly used to solve Sudoku puzzles, various alternative methods like harmony search, heuristics, and genetic algorithms have been evaluated, though they often encounter failures necessitating restarts. Thus, this paper seeks to apply graph theory to devise a more efficient and failproof Sudoku-solving algorithm relative to the naive approaches. We developed an algorithm and implemented it to generate 10,000 Sudoku puzzles of varying difficulty, subjecting them to tests with both conventional methods and our graph coloring approach. Our technique demonstrated no failures in practical iterations. Results indicate that this approach boasts a narrower spread, and shorter median time, and proves sufficient to solve complex Sudoku puzzles with a relatively limited number of moves.

2 Introduction

Sudoku is a logic-based game. Typically, in Sudoku, the board starts out as a 9 by 9 grid with a few pre-filled boxes containing digits between 1 and 9. The player's task is to then assign each box a digit between 1 and 9 ensuring that every column, row, each 3 by 3 subgrid contains each digit exactly once. Other variations of the game include different grid sizes and/or restrictions. Solving Sudoku puzzles essentially boils down to extrapolating a large amount of information from a modest set of clues or constraints. Various solving methods exist — the most popular being brute force and backtracking — though our algorithm leverages graph theory to enhance solving time and consistency. Beyond mere puzzle-solving, our Sudoku-solving algorithm can be utilized for steganography, which deals with hiding data or information. For example, image steganography hides data within a cover image. The data that must be hidden corresponds to the few numbers given at the start of the Sudoku puzzle, while the cover image including this data corresponds to the solved Sudoku board. Like this, cover images can be generated via Sudoku-solving algorithms. Similarly, audio steganography conceals information within audio files, which can protect sounds from being copied illegitimately. This versatility extends to practical domains, developing school schedules for each individual within a large group. The restrictions include the classes each individual must take and limits on class size, from which Sudoku-solving algorithms can then create schedules for teachers and students that remain within these restrictions.

3 Methodology

3.1 Existing Approaches

3.1.1 Naive Approaches

The simplest method involves brute force – randomly assigning numbers to the puzzle board, and checking if this works. Clearly, this method is very time-consuming. As such, the most common method of Sudoku is an adaptation of this, creating a search space for the puzzle and applying a depth-first search with backtracking.

3.1.2 Evolutionary and Genetic Algorithms

Sato [1] proposes crossover and mutation operators that preserve building blocks by using local search, for use in a genetic algorithm to solve Sudoku. 100 runs were performed with a success rate of 100% for the easy and intermediate problems, 96% for the difficult problems, 98 %, 83%, and 58% for the three example problems.

3.1.3 Harmony Search

Geem [2] evaluates harmony search as a means of solving Sudoku, emulating different behaviors of musicians including random play, memory-based play and pitch-adjusted play. The harmony algorithm was able to solve an easy Sudoku puzzle in 9 seconds, though unable to solve more difficult ones.

3.1.4 Heuristics

Jones et al. [3] implies the use of heuristics for search-based approaches to solve sudoku problems, employed in a modified steepest ascent hill-climbing algorithm. All runs eventually succeeded. However, more difficult runs required multiple restarts, and the algorithm itself ran extremely slowly.

3.2 Graph Theory Approach

Graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices, which represent the objects, and edges, which represent the relations between the objects [4]. Graph coloring is a problem in graph theory in which colors are assigned to the vertices of a graph so that no two adjacent, or connecting, vertices have the same color.

We can apply this principle to Sudoku, where the Sudoku puzzle can be viewed as a graph with each box represented as a vertex. Each vertex is connected to every other vertex in the same row and column, creating edges between such. Additionally, we create connections between each vertex and every other vertex in its 3x3 subgrid.

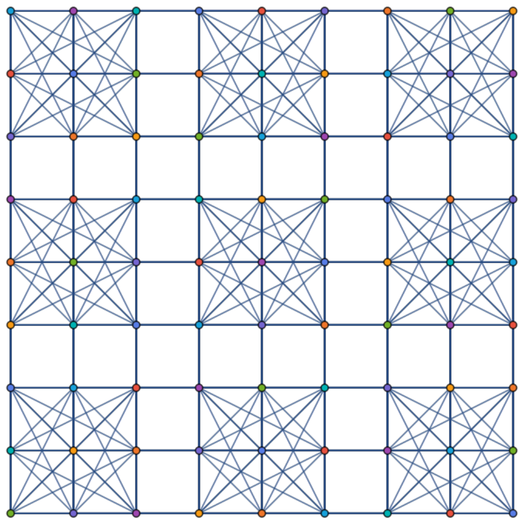

Figure 1: Connections of a graphical representation of a Sudoku grid

Thus, we can see that solving a Sudoku puzzle is equivalent to creating a graph coloring for Figure 1 with 9 different colors - one for each number.

So, we have established that graph coloring is a valid approach to Sudoku puzzles, and on paper, there is no foreseeable possibility of failures (as opposed to heuristics when in local optima). Thus, we will compare this process with those most alike - brute force and backtracking.

3.3 Graph Coloring Algorithm Implementation

To simulate this process with code, we can represent the numbers in the sudoku board as an 81-element square matrix. The elements of this matrix represent the vertices of our graph. We can use a hashmap to map connections between vertices, which is equivalent to simulating an edge on the graph.

Next, we design connections to account for the rules of Sudoku. As discussed in 3.2, we map connections between vertices and other vertices in their row, column, and corresponding 3x3 subgrid.

Finally, we can implement a recursive graph coloring algorithm: to determine the coloring of a specific vertex, we look at the coloring of every connected vertex. After determining all colors of the 81-element matrix, we can then map them back to be values of the solved sudoku board.

We compared this algorithm to others by testing their performances on 10000 randomly generated sudoku puzzles and measuring their runtime.

4 Results

Solving Method	Avg Time Taken (s)	Standard Deviation (s)
Graph Coloring	0.00788	0.00080
Backtracking	0.01211	0.00207
Brute Force	0.01437	0.00113

Figure 2: Avg Time Taken and Standard Deviations of Various Sudoku Solving Methods

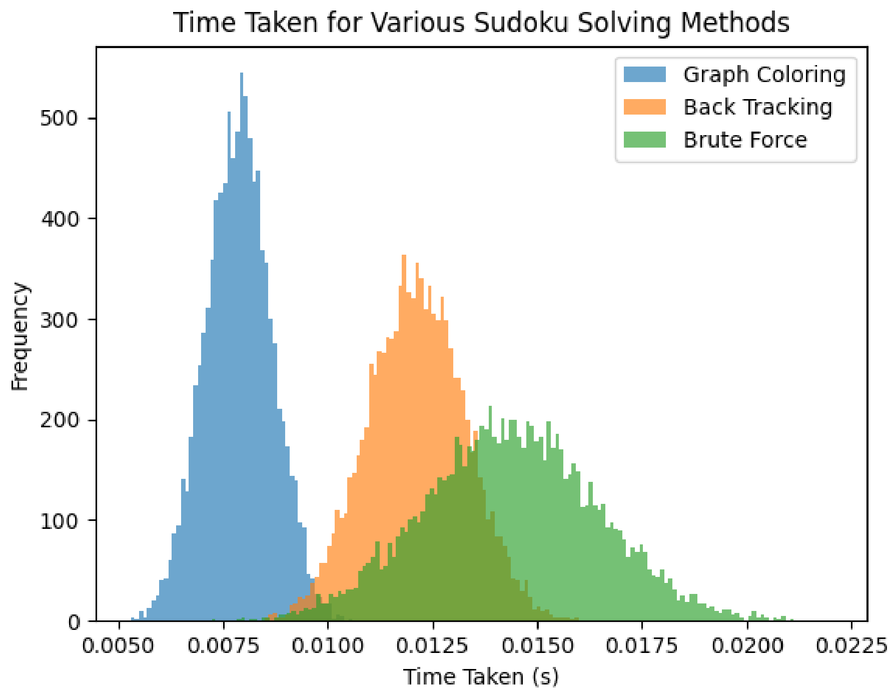

Figure 3: Graphical Representation of Solving Times for Various Sudoku Solving Methods (10000 Repetitions)

As shown in Figure 2, the average time taken to solve a Sudoku puzzle with the graph coloring method is significantly lower than with the other two methods. This is even more evident in the histograms of Figure 3. The backtracking and brute force methods have solving times that form relatively comparable distributions and centers. In contrast, the graph coloring method is clearly more efficient and has a significantly lower center than the other two.

5 Conclusion

This report demonstrates how graph coloring can be utilized to solve Sudoku puzzles. The main benefit of this algorithm is its decreased average time and standard deviation relative to existing technique, proving its increased speed and consistency respectively. In the future, our proposed method can be implemented to improve the efficiency of schedule creation and steganography.

6 Appendix

6.1 References

[1] Sato, Y. 2010. Solving Sudoku with Genetic Operations that Preserve Building Blocks. In Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG '10), 23-29.

[2] Geem, Z.W. 2007. Harmony Search Algorithm for Solving Sudoku. In Proceedings of KES 2007, 371-378.

[3] Jones, S. K., Roach, P. A., & Perkins, S. (2007). Construction of Heuristics for a Search-Based Approach to Solving Sudoku. International Conference on Innovative Techniques and Applications of Artificial Intelligence.

[4] Keller, M. T., & Trotter, W. T. (2017). Applied combinatorics. Mitchel T. Keller, William T. Trotter

6.2 Code

All of the code files used for the graph coloring algorithm and simulation can be found here: `https://github.com/GeorgeLiu59/SudoKolor`

The code for the brute force and backtracking solving methods can be found here: `https://www.geeksforgeeks.org/sudoku-backtracking-7`/