Implementing Backtracking for Sudoku Solver

Chirayu Agrawal
Computer Science and Engineering
Shiv Nadar University
Delhi-NCR,India
ca501@snu.edu.in

Kunal Passan
Computer Science and Engineering
Shiv Nadar University
Delhi-NCR,India
kp728@snu.edu.in

Diva Pandey
Computer Science and Engineering
Shiv Nadar University
Delhi-NCR,India
dp631@snu.edu.in

***Abstract*—Sudoku is a widely recognized combinatorial puzzle that requires filling a 9x9 grid with numbers such that each row, column, and 3x3 subgrid contains all digits from 1 to 9, without repetition. The problem of solving Sudoku can be modeled as a constraint satisfaction problem (CSP), making it suitable for a variety of algorithmic approaches. In this paper, we explore the efficacy of the backtracking algorithm in solving Sudoku puzzles of varying difficulty levels. Backtracking is a depth-first search algorithm that incrementally builds candidates to the solutions and abandons candidates ("backtracks") as soon as it determines that a candidate cannot possibly lead to a valid solution.Additionally, we incorporate OpenCV for image processing to detect and extract Sudoku grids from images, converting the visual data into a digital format suitable for algorithmic solving. This combination of image detection and backtracking creates a seamless pipeline for automatically solving Sudoku puzzles from real-world inputs.**

***Index Terms*—Sudoku, backtracking, opencv, image detection,constraint satisfaction problem**

I. Introduction

Over the decades, puzzle-solving has remained a source of fascination, with Sudoku emerging as one of the most captivating brain-teasers. Widely featured in newspapers, magazines, and mobile apps, Sudoku originated in Japan in 1984 and quickly became a global phenomenon. Today, more than 60,000 Sudoku magazines are published monthly in Japan alone, catering to enthusiasts across all age groups. The game's widespread appeal has led to its digital transformation, allowing users to solve puzzles through various apps and online platforms.

With the growing shift towards technology-driven entertainment, many applications now offer randomly generated Sudoku puzzles alongside pre-uploaded solutions. Building on this trend, we propose a method to obtain images of Sudoku puzzles, detect them from images using advanced image processing techniques, and solve them efficiently using backtracking algorithms. Unlike deep learning methods, which require extensive training on large datasets and significant computational resources, our approach leverages classical algorithms to solve puzzles in real-time while maintaining accuracy.

In this paper, we employ OpenCV, a leading library in computer vision, for contour detection and Optical Character Recognition (OCR) to extract the puzzle grid from images. The extracted puzzle is then solved using backtracking, a depth-first search algorithm that systematically explores possible solutions. This method offers a lightweight, efficient solution for solving Sudoku puzzles detected from images without the need for extensive model training, providing quick and reliable results.

II. Literature Survey

A. Image Detection Using OpenCV

OpenCV is widely used for detecting and extracting Sudoku grids from images through techniques like contour detection and edge detection. Chien and Kuo (2017) employed adaptive thresholding to improve grid extraction from varying image qualities, while Patel et al. (2019) enhanced grid detection with perspective transformation to correct distortion, forming a reliable foundation for image-based Sudoku detection.

B. Optical Character Recognition (OCR) for Digit Extraction

OCR, particularly Tesseract, is commonly used for extracting Sudoku digits from images. Krishnan et al. (2020) demonstrated how preprocessing techniques, such as noise reduction and contrast enhancement via OpenCV, improved OCR accuracy. However, challenges persist with handwritten digits and low-quality images, prompting the need for custom OCR solutions.

C. Sudoku Solving Algorithms

Backtracking has been a favored method for solving Sudoku due to its depth-first search approach. Simonis (2005) established its effectiveness for constraint satisfaction problems like Sudoku. Recent comparisons, such as Smith et al. (2021), found that while heuristic methods can be faster, backtracking remains reliable, especially with constraint propagation to reduce the search space.

D. Combining Image Detection and Solving

Lee et al. (2018) integrated OpenCV, Tesseract OCR, and backtracking to create an automated Sudoku solver with a 95% accuracy rate. Their system demonstrated that classical methods can effectively solve Sudoku from images without relying on deep learning, making it resource-efficient.

E. Limitations and Future Work

Challenges remain in improving OCR accuracy and handling varying image quality. Future work could explore better preprocessing techniques and hybrid approaches that combine algorithmic solvers with machine learning for improved performance across different conditions.

III. Methodology

A. System Overview

The proposed system for detecting and solving Sudoku puzzles consists of three main components: image preprocessing, grid detection, digit recognition, and Sudoku solving. The system takes an input image of a Sudoku puzzle, processes it to extract the grid, recognizes the digits within the grid, and then applies a backtracking algorithm to solve the puzzle. The output is a solved Sudoku grid, displayed alongside the original image for comparison.

B. Image Preprocessing

The first step in the system is image preprocessing, where OpenCV is utilized to enhance the quality of the input image. Techniques such as adaptive thresholding are employed to convert the image into a binary format, effectively distinguishing the puzzle grid and its digits from the background. Noise reduction filters are applied to minimize distortions caused by lighting variations or image artifacts, ensuring clearer contours for further analysis.

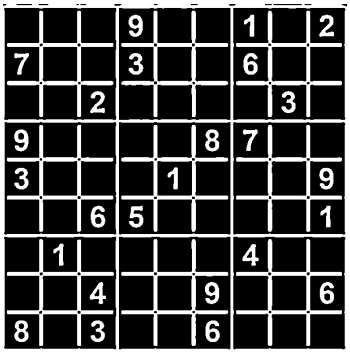

Fig. 1. Binary image after preprocessing with adaptive thresholding

C. Grid Detection

Once the image is preprocessed, the system proceeds to detect the Sudoku grid. OpenCV's contour detection method identifies the largest contour, which typically corresponds to the outer boundary of the Sudoku grid. Perspective transformation is applied to correct any distortions in the grid, allowing for accurate extraction of the 9x9 cell structure. This process involves calculating the four corner points of the grid and warping the image to achieve a top-down view.

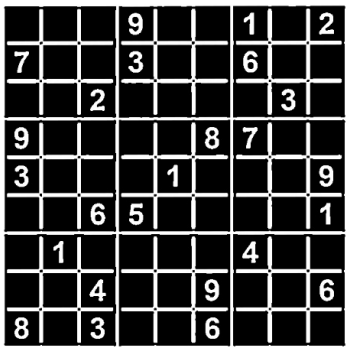

Fig. 2. Top-down view of the Sudoku grid after perspective transformation

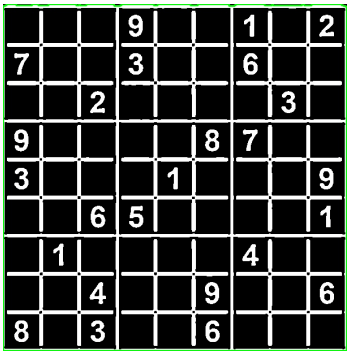

Fig. 3. Conotur detection

D. Digit Recognition using OCR

After extracting the grid, Optical Character Recognition (OCR) is employed to identify the digits within each cell. Tesseract OCR is utilized for this purpose, leveraging its capabilities to convert the visual representations of digits into machine-readable text. Preprocessing steps, such as enhancing contrast and applying binarization, are essential to improve OCR accuracy. If OCR fails to recognize any digits due to low image quality or handwriting variations, fallback mechanisms are implemented to alert the user or request a clearer image.

Fig. 4. a single digit recognised after OCR

E. Sudoku Solving using Backtracking

In the Sudoku solving process, once the digits are recognized and placed into a 2D array (which represents the Sudoku grid), a backtracking algorithm is used to systematically attempt to solve the puzzle. Backtracking is a recursive algorithm, often compared to depth-first search (DFS), as it explores possible solutions by going deep into a sequence of choices and backtracking when a solution path fails.

Here's how the backtracking algorithm works in more detail:

1. Identify Empty Cells: The algorithm begins by finding the first empty cell (denoted by a 0 or empty space). This is

the starting point, where the algorithm will attempt to place a valid digit.

2. Try Digits Sequentially: For the selected empty cell, the algorithm tries placing each number from 1 to 9 sequentially. For each number placed, it checks if that number complies with the rules of Sudoku: - The number should not already exist in the same row. - The number should not already exist in the same column. - The number should not already exist in the 3x3 subgrid.

3. Validation Check: If a number meets all these conditions, it is considered a ”safe” placement. The algorithm proceeds to the next empty cell and repeats the process.

4. Backtracking: If a conflict arises (meaning a valid placement for a number cannot be found in the current cell), the algorithm backtracks. This involves removing (or ”unplacing”) the last number added to the grid and trying the next number in that cell. The process continues, backtracking multiple steps if necessary, until a safe number is found.

5. Recursive Exploration: The algorithm uses recursion to ”go deep” by filling each cell in sequence. It explores potential solutions in a depth-first manner, backtracking whenever it reaches a dead end.

6. Complete or Unsatisfiable: If the algorithm successfully fills every cell without conflicts, the puzzle is solved. If the algorithm returns to the starting cell without finding a valid solution, it concludes that the puzzle is unsolvable.

F. Output

The final output consists of the solved Sudoku grid displayed alongside the original image of the puzzle. The solved digits can be overlaid on the original image for visual clarity, highlighting the changes made during the solving process. This feature allows users to compare the input puzzle with the solution easily, enhancing the overall user experience.

IV. Output Results

The results of the Sudoku solving process are illustrated in the figures below. The first figure presents the original Sudoku puzzle as input, while the second figure displays the solution overlaid on the original image.

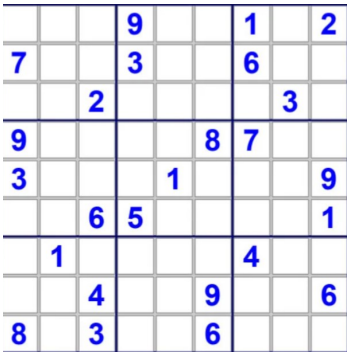

Fig. 5. Original Sudoku Puzzle

Fig. 6. Solved Sudoku Puzzle Overlaid on Original Image

V. Results

The performance of the backtracking algorithm was evaluated by testing it on Sudoku puzzles of varying difficulty levels. The runtime of the algorithm was measured specifically for solving the Sudoku board after the digit recognition and preprocessing steps. The results are presented in Table I.

TABLE I
Runtime of the Backtracking Algorithm Across Different Difficulty Levels

Difficulty Level	Runtime (seconds)
Easy	0.0010
Medium	0.0030
Hard	0.0110

As shown in Table I, the backtracking algorithm solved easy puzzles in approximately 0.0010 seconds, medium puzzles in 0.0030 seconds, and hard puzzles in 0.0110 seconds. The increase in runtime with puzzle difficulty is consistent with the expected behavior of backtracking algorithms, as more difficult puzzles require deeper recursive exploration to satisfy Sudoku constraints.

These results demonstrate that the algorithm can solve easy and medium puzzles almost instantly, while hard puzzles require marginally more processing time, though still well within acceptable limits for real-time applications. This makes the backtracking approach suitable for a wide range of Sudoku-solving scenarios, balancing performance and simplicity in implementation.

VI. Limitations

While the proposed Sudoku solver demonstrates effective performance across varying difficulty levels, several limitations warrant consideration:

A. Dependence on Image Quality

The algorithm’s performance heavily relies on the quality of the input image. Blurred, distorted, or poorly lit images may hinder accurate digit recognition and grid detection, leading to incorrect results.

B. Limited Character Recognition

The Tesseract OCR engine used for digit recognition may misidentify characters in cases of low contrast, overlapping digits, or unusual handwriting styles. This limitation can result in incorrect entries in the Sudoku board, ultimately affecting the solver's accuracy.

C. Fixed Grid Size

The algorithm is designed specifically for 9x9 Sudoku puzzles. Adapting the code to handle different grid sizes (e.g., 4x4, 16x16) would require substantial modifications, including changes in digit extraction and validation logic.

D. Backtracking Limitations

The backtracking algorithm, while effective, may struggle with particularly complex Sudoku puzzles. Although performance was satisfactory for the tested cases, its efficiency could degrade significantly with more challenging configurations, leading to increased runtimes.

VII. Improvements

To enhance the capabilities and performance of the Sudoku solver, several improvements can be implemented:

A. Advanced Image Processing Techniques

Incorporating techniques such as image enhancement, edge detection, or contour filtering could improve grid detection and digit recognition. These enhancements would help mitigate the impact of poor image quality on the overall performance of the algorithm.

B. Custom OCR Models

Training a custom OCR model specifically for recognizing digits in Sudoku puzzles could significantly reduce errors associated with character recognition. Utilizing a dataset of labeled Sudoku digits can enhance the accuracy of digit extraction from the grid.

C. Dynamic Grid Size Handling

Refactoring the code to support dynamic grid sizes would increase the versatility of the solver. Implementing a mechanism to detect and process various Sudoku configurations would make the application more user-friendly.

D. Hybrid Solving Approaches

Combining backtracking with other solving techniques, such as constraint propagation or heuristics, could improve the efficiency of the algorithm, especially for complex puzzles. Implementing a hybrid approach could reduce runtime and enhance the solver's ability to handle more challenging configurations.

E. User Interface Development

Developing a graphical user interface (GUI) would enhance user experience by allowing users to easily upload images and view solutions. A GUI could also provide options for selecting puzzle difficulty and visualizing the solving process step-by-step.

VIII. References

1) C. Kaundilya, D. Chawla, and Y. Chopra, "Automated Text Extraction from Images using OCR System," in *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*, 2019.
2) A. K. Maji and R. K. Pal, "Sudoku Solver Using Minigrid-Based Backtracking," in *2014 IEEE International Advance Computing Conference (IACC)*, 2014.
3) C. Patel, A. Patel, and D. Patel, "Optical Character Recognition by Open Source OCR Tool Tesseract: A Case Study," *International Journal of Computer Applications*, vol. 55, no. 10, Oct. 2012.
4) J. R. Bitner and E. Reingold, "Backtracking Programming Techniques," *Communications of the ACM*, 1975.
5) P. Nikitha, T. Nikitha, M. Nikshitha, T. Nikhil, T. Nithin, K. Nithish Reddy, and Prof. S. J. Hamilpure, "Object Detection Using OpenCV with Python," *International Research Journal of Modernization in Engineering Technology and Science*.
6) A. Narayanaswamy, Y. P. Ma, and P. Shrivastava, "Image Detection and Digit Recognition to Solve Sudoku as a Constraint Satisfaction Problem."
7) S. Hira, N. Bhagwatkar, K. Agrawal, and N. Loya, "Sudoku Solver: A Comparative Study of Different Algorithms and Image Processing Techniques."