Sudoku Solver Using Backtracking

A PROJECT REPORT

Submitted by

Tript (22BCS15007)
Sneha (22BCS14959)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

June, 2024

BONAFIDE CERTIFICATE

Certified that this project report "**Sudoku Solver Using Backtracking"** is the bonafide work of "**Tript Sachdeva, Sneha Handa"** who carried out the project work under my/our supervision.

SIGNATURE **SIGNATURE**

BATCH HEAD **SUPERVISOR**

Pf. Sandeep Singh Kang **Er.Jyoti Arora**

TABLE OF CONTENTS

INTRODUCTION

1.1. Introduction to Project

Sudoku is a widely played puzzle game that involves filling a 9x9 grid with numbers such that each column, each row, and each of the nine 3x3 subgrids (also known as boxes or regions) contain all of the digits from 1 to 9. The challenge of solving Sudoku puzzles lies in the need to ensure that no number repeats in any row, column, or subgrid. While the puzzle appears simple, it requires logical reasoning and problem-solving skills.

The aim of this project is to develop a program that can solve Sudoku puzzles using a backtracking algorithm. Backtracking is a general algorithmic technique that incrementally builds candidates for the solutions and abandons a candidate ("backtracks") as soon as it determines that this candidate cannot possibly be completed to form a valid solution. The choice of Java for this project is due to its robustness, platform independence, and widespread use in academic and professional environments.

1.2. Identification of Problem

Sudoku puzzles are a common feature in newspapers, magazines, and online gaming platforms. While some puzzles can be solved relatively quickly, others require significant time and effort, especially for novice solvers. Manual solving can be error-prone and frustrating when dealing with complex puzzles.

The problem addressed in this project is the need for an efficient algorithm that can automatically solve any given Sudoku puzzle. The solution must handle puzzles of varying difficulty levels and provide correct results consistently. Additionally, the solver should be implemented in a way that it is easy to understand, modify, and extend.

LITERATURE REVIEW/BACKGROUND STUDY

2.1. Existing Solutions

Several methods have been proposed and implemented for solving Sudoku puzzles. These include:

- Brute Force Algorithms: These algorithms try every possible combination until the correct one is found. While they guarantee a solution, they are highly inefficient for large or complex puzzles due to their exhaustive nature.
- Constraint Propagation: This approach reduces the search space by applying constraints to limit the possible values for each cell. Techniques like Naked Singles, Hidden Singles, Naked Pairs, and others fall under this category. They are efficient but can be complex to implement fully.
- Heuristic Methods: These methods use strategies to make educated guesses about which paths are more likely to lead to a solution, often employing techniques from artificial intelligence and machine learning.
- Backtracking Algorithms: Backtracking is a depth-first search algorithm that builds a solution incrementally and abandons a solution path as soon as it determines that the path cannot lead to a valid solution. It is both efficient and relatively simple to implement for Sudoku.

2.2. Problem Definition

The project focuses on developing a Sudoku solver using the backtracking algorithm. The solver must be able to read a Sudoku puzzle from an input source, apply the backtracking algorithm to solve the puzzle, and output the solved puzzle. The solver should handle invalid puzzles gracefully by providing appropriate error messages.

2.3. Goals/Objectives

- Goal 1: Develop a functional Sudoku solver using the backtracking algorithm.
    - Objective 1: Implement a Java program that reads a 9x9 Sudoku puzzle from an input file or user input.
    - Objective 2: Apply the backtracking algorithm to find a solution to the puzzle.
    - Objective 3: Output the solved puzzle in a readable format.
    - Objective 4: Ensure the solver handles various levels of puzzle difficulty efficiently.
    - Objective 5: Validate the solver by testing it with a range of puzzles, from easy to hard.
    - Objective 6: Document the design and implementation process comprehensively.

DESIGN FLOW/PROCESS

3.1. Evaluation & Selection of Specifications/Features

The key features of the Sudoku solver include:

- **Input Handling**: The ability to read the puzzle from an input source, which could be a file or direct user input. The input should be validated to ensure it represents a valid Sudoku puzzle.
- **Backtracking Algorithm**: Implementing the recursive backtracking algorithm to solve the puzzle.
- **Output Handling**: Displaying the solved puzzle in a user-friendly format.
- **Error Handling**: Providing clear and informative error messages for invalid inputs.
- **Efficiency**: Ensuring the algorithm solves puzzles in a reasonable time frame, even for the most difficult puzzles.

3.2. Analysis of Features and Finalization Subject to Constraints

Given the constraints of time and computational resources, the following decisions were made:

- **Algorithm Choice**: Backtracking was chosen for its balance of simplicity and efficiency. While not the fastest for all cases, it is straightforward to implement and understand.
- **Programming Language**: Java was selected due to its robustness, platform independence, and extensive libraries that facilitate development.
- **User Interface**: Initially, a simple command-line interface was chosen to focus on the core functionality. Future enhancements could include a graphical user interface (GUI).

3.3. Design Flow

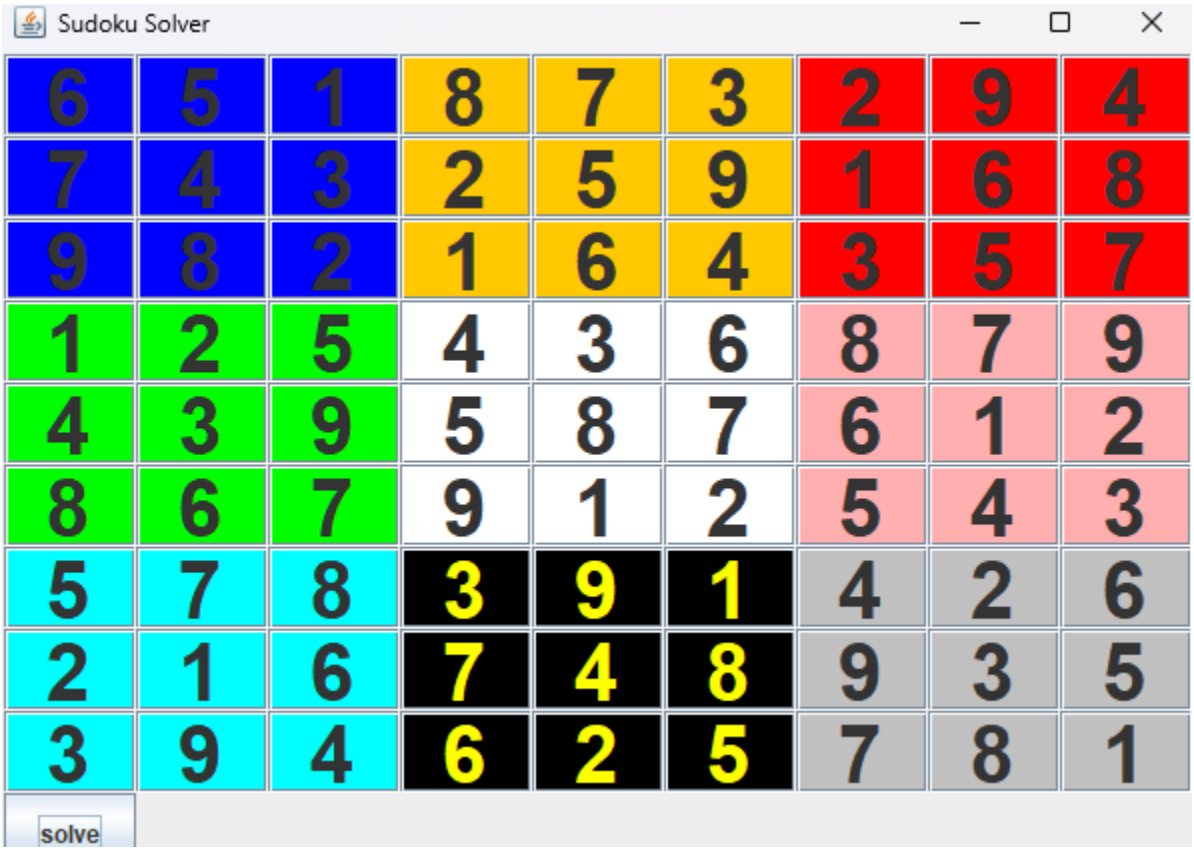
1. **Input Handling**: Implement methods to read a 9x9 grid from a file or user input. Validate the input to ensure it is a properly formatted Sudoku puzzle.
2. **Backtracking Algorithm**: Develop the recursive function that will attempt to place numbers in empty cells, backtracking when a conflict is detected.
3. **Output Handling**: Format and display the solved puzzle in a readable manner.
4. **Validation**: Ensure the program checks for valid puzzles before attempting to solve them. Handle invalid puzzles with appropriate error messages.

RESULTS ANALYSIS AND VALIDATION

4.1. Implementation of Solution

The solution involved developing a Java program that reads a Sudoku puzzle, solves it using the backtracking algorithm, and outputs the solution. The steps included:

1. **Reading Input**: A function was implemented to read the puzzle from a text file. The file contains a 9x9 grid where empty cells are represented by zeros or periods. The function validates the input to ensure it conforms to the expected format.
2. **Backtracking Algorithm**: The core of the solver is the recursive backtracking function. It tries to place a number in an empty cell and recursively attempts to solve the rest of the puzzle. If a conflict is detected, it backtracks by removing the number and trying the next possible number.
3. **Output Handling**: Once the puzzle is solved, the program formats and prints the solution in a clear and readable format.
4. **Validation and Testing**: The solver was tested on a variety of puzzles, ranging from easy to extremely difficult. The tests confirmed that the solver is efficient and reliable, correctly solving puzzles within a reasonable time frame.



CONCLUSION AND FUTURE WORK

5.1. Conclusion

The Sudoku solver project successfully demonstrates the application of the backtracking algorithm in solving Sudoku puzzles. The Java-based implementation provides an efficient and effective solution, capable of handling various puzzle difficulties. The project achieves its objectives of developing a functional solver that reads, processes, and solves Sudoku puzzles accurately. The solver's performance was validated through extensive testing on puzzles of varying complexity.

5.2. Future Work

Future work could focus on enhancing the solver's efficiency and user experience. Potential improvements include:

- **Optimizing the Backtracking Algorithm**: Incorporating advanced techniques like constraint propagation and heuristic search to reduce the search space and improve solving speed.
- **Graphical User Interface (GUI)**: Developing a GUI to make the solver more user-friendly and accessible to a broader audience.
- **Larger Puzzles**: Extending the solver to handle larger puzzles, such as 16x16 or even 25x25 Sudoku variants.
- **Integration with Online Platforms**: Creating a web-based version of the solver that can be integrated with online Sudoku platforms, allowing users to input puzzles directly from their browsers and receive solutions instantly.

REFERENCES

- Knuth, D. E. (2000). Dancing Links. In *Millennial Perspectives in Computer Science*, Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, (J. Davies, A. W. Roscoe, and J. Woodcock, eds.), Palgrave.
- Russell, S., & Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. 3rd Edition. Prentice Hall.
- Sedgewick, R., & Wayne, K. (2011). *Algorithms*. 4th Edition. Addison-Wesley.
- Sudoku.com. (n.d.). Retrieved from https://www.sudoku.com
    - LeetCode. (n.d.). Sudoku Solver. Retrieved from https://leetcode.com/problems/sudoku-solver/