SUDOKU SOLVER(BACKTRACKING)

A PROJECT REPORT

Submitted by

UMA DEVI S (8115U23EC115)

in partial fulfillment of requirements for the award of the course

EGA1121 – DATA STRUCTURES
in

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF ENGINEERING

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

MAY 2024

K.RAMAKRISHNAN COLLEGE OF ENGINEERING
(AUTONOMOUS) SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report titled "**SUDOKU SOLVER (BACKTRACKING)**" is the bonafide work of **UMA DEVI S(8115U23EC115)** , who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported here in does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE	**SIGNATURE**
Dr.T.M.NITHYA M.E., Ph.D.,	**Mrs.S.RAJESWARI M.E.,**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
ASSOCIATE PROFESSOR	**ASSISTANT PROFESSOR**
DEPARTMENT OF CSE	**DEPARTMENT OF CSE**
K.RAMAKRISHNAN COLLEGE OF ENGINEERING	**K.RAMAKRISHNAN COLLEGE OF ENGINEERING**
(AUTONOMOUS)	**(AUTONOMOUS)**
SAMAYAPURAM–621112.	**SAMAYAPURAM–621112.**

Submitted for the end semester examination held on …………….

DECLARATION

I jointly declare that the project report on **"SUDOKU SOLVER (BACKTRACKING)"** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI"** for the requirement of Degree of BACHELOR OF ENGINEERING. This project report is submitted on the partial fulfillment of the requirement of the award of the course **EGA1121- DATA STRUCTURES**

Signature

UMA DEVI S

Place: Samayapuram
Date:

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and indebtedness to our institution, "**K.RAMAKRISHNAN COLLEGE OF ENGINEERING (Autonomous)**", for providing us with the opportunity to do this project.

I extend our sincere acknowledgment and appreciation to the esteemed and honorable Chairman, **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding our project and offering an adequate duration to complete it.

I would like to thank **Dr. D. SRINIVASAN, M.E., Ph.D., FIE., MIIW., MISTE., MISAE., C. Engg.,** Principal, who gave the opportunity to frame the project to full satisfaction.

I thank **Dr.T.M.NITHYA., M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for providing her encouragement in pursuing this project.

I wish to convey our profound and heartfelt gratitude to our esteemed project guide **Mrs.S.RAJESWARI ., M.E.,** Department of Computer Science and Engineering, for her incalculable suggestions, creativity, assistance and patience, which motivated us to carry out this project.

I render our sincere thanks to the Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

INSTITUTE VISION AND MISSION

VISION OF THE INSTITUTE:

To achieve a prominent position among the top technical institutions.

MISSION OF THE INSTIITUTE:

M1: To bestow standard technical education par excellence through state of the art infrastructure, competent faculty and high ethical standards.

M2: To nurture research and entrepreneurial skills among students in cutting edge technologies.

M3: To provide education for developing high-quality professionals to transform the society.

DEPARTMENT VISION AND MISSION

VISION OF THE DEPARTMENT:

To create eminent professionals of Computer Science and Engineering by imparting quality education.

MISSION OF THE DEPARTMENT:

M1: To provide technical exposure in the field of Computer Science and Engineering through state of the art infrastructure and ethical standards.

M2: To engage the students in research and development activities in the field of Computer Science and Engineering.

M3: To empower the learners to involve in industrial and multi-disciplinary projects for addressing the societal needs.

PROGRAM EDUCATIONAL OBJECTIVES (PEOs):

Our graduates shall,

PEO1: Analyze, design and create innovative products for addressing social needs.

PEO2: Equip themselves for employability, higher studies and research.

PEO3: Nurture the leadership qualities and entrepreneurial skills for their successful career

PROGRAM SPECIFIC OUTCOMES (PSOs):

Students will be able to,

PSO1: Apply the basic and advanced knowledge in developing software, hardware and firmware solutions addressing real life problems.

PSO2: Design, develop, test and implement product-based solutions for their career enhancement.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

This C program efficiently solves Sudoku puzzles using a backtracking algorithm. It features several key components: a grid printing function (printGrid), which outputs the current state of the Sudoku grid; a safety check function (isSafe), which ensures a number can be placed in a cell without breaking Sudoku rules by checking the corresponding row, column, and 3x3 subgrid; and a backtracking solver function (solveSudoku), which attempts to fill empty cells with numbers 1 through 9, recursively solving the puzzle and backtracking as necessary when a dead end is reached. The main function ties everything together, prompting the user to input the Sudoku grid, invoking the solver, and printing the solved grid if a solution is found. If the puzzle is unsolvable, it informs the user. The program systematically fills the grid, ensuring each placement is valid, and uses backtracking to explore all possible configurations until it finds a solution. So this program will efficiently solve the problem.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

UI	-	User Interface
CLI	-	Command Line Interface
GUI	-	Graphical User Interface
CSP	-	Constrain Satisfaction Program
GUI	-	Graphical User Interface

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

This C program is designed to solve Sudoku puzzles using a backtracking algorithm. It takes a 9x9 grid as input, where empty cells are represented by zeros, and systematically attempts to fill in the grid by checking each cell's validity according to Sudoku rules. The program includes functions to print the grid, check the safety of number placements, and solve the puzzle through recursive backtracking. Once the puzzle is solved, it outputs the completed grid; if no solution exists, it notifies the user. This approach ensures that all possible configurations are explored to find a valid solution.

1.2 PURPOSE AND IMPORTANCE

The purpose of this Sudoku solver program is to automate the process of solving Sudoku puzzles using a backtracking algorithm. It is an educational tool that demonstrates the practical application of recursive problem-solving and constraint satisfaction techniques. This program is important because it helps users understand the backtracking algorithm, a fundamental concept in computer science, through a real-world example. It enhances problem-solving skills by illustrating how to handle complex, constraint-based problems and provides a foundation for more advanced topics such as optimization and game theory. Additionally, it shows the practical utility of algorithmic solutions in everyday tasks, making it a valuable resource for both learning and application.

The importance of this Sudoku solver program lies in its demonstration of key computer science principles and practical applications. Firstly, it provides a clear example of the backtracking algorithm, which is essential for solving constraint satisfaction problems found in various fields, including artificial intelligence and operations research. Secondly, it enhances problem-solving skills by illustrating how to approach and resolve complex puzzles through systematic trial and error. Additionally, it serves as a foundational tool for understanding more advanced concepts such as recursion, grid traversal, and constraint checking. The program's practical application in solving Sudoku puzzles showcases the real-world utility of algorithmic solutions, making it a valuable educational resource and a useful tool for both learners and enthusiasts

1.3 OBJECTIVES

1. Automate the solving process of 9x9 Sudoku puzzles.
2. Demonstrate the implementation and efficiency of the backtracking algorithm.
3. Serve as an educational tool for understanding recursion and constraint satisfaction.
4. Improve logical thinking and problem-solving skills.
5. Provide a practical application of computer science principles.
6. Lay the foundation for understanding more complex algorithms

1.4 PROJECT SUMMARIZATION

The Sudoku Solver is a C program designed to automate the solving process of 9x9 Sudoku puzzles. It utilizes a backtracking algorithm to systematically fill in empty cells while ensuring that each placement follows Sudoku rules. The program includes functions for printing the grid, checking the validity of number placements, and recursively solving the puzzle. The main function prompts the user to input the Sudoku grid, invokes the solver, and outputs the completed puzzle or notifies if no solution exists. This project serves as an educational tool for understanding recursion, constraint satisfaction, and algorithmic problem-solving. It enhances logical thinking skills and demonstrates practical applications of computer science principles. Additionally, it lays the foundation for learning more complex algorithms and optimization techniques

CHAPTER 2

PROJECT METHODOLOGY

2.1 INTRODUCTION TO SYSTEM ARCHITECTURE

System architecture refers to the conceptual model that defines the structure, behaviour, and more views of a system. It serves as a blueprint for both the system and the project developing it. In the context of the SUDOKU SOLVER (BACKTRACKING) , the system architecture outlines how various components interact and work together to achieve the desired functionality.

2.1.1 High-Level System Architecture

The high-level system architecture for the phone directory application typically consists of several key components:

(i) User Interface (UI)
(ii) Application Logic
(iii) Data Management Layer

2.1.2 Components of the System Architecture

a. User Interface (UI)

The User interface for the Sudoku solver program presents clear instructions for inputting the Sudoku puzzle and understanding the format. Users are prompted to input the puzzle, with '0' representing empty cells, row by row and separated by spaces. Upon input completion, the program displays the solution if found, formatted in a 9x9 grid, or notifies the user if no solution exists. This interface streamlines the user experience, guiding them through the input process and providing immediate feedback on the puzzle's solution status.

b. Application Logic

The Application logic revolves around recursively solving the Sudoku puzzle using backtracking. It handles user input, checks the validity of number placements, backtracks when necessary, and prints the solution if found

c. Data Management Layer

The Data Management Layer is responsible for managing the data structures used to store and retrieve contact information efficiently.

2.2 DETAILED SYSTEM ARCHITECTURE DIAGRAM

Include a diagram that visually represents the system architecture. The diagram should depict how each component interacts with the others. For example, it can show the User Interface sending requests to the Application Logic, which in turn interacts with the Data Management Layer and the Storage Layer.

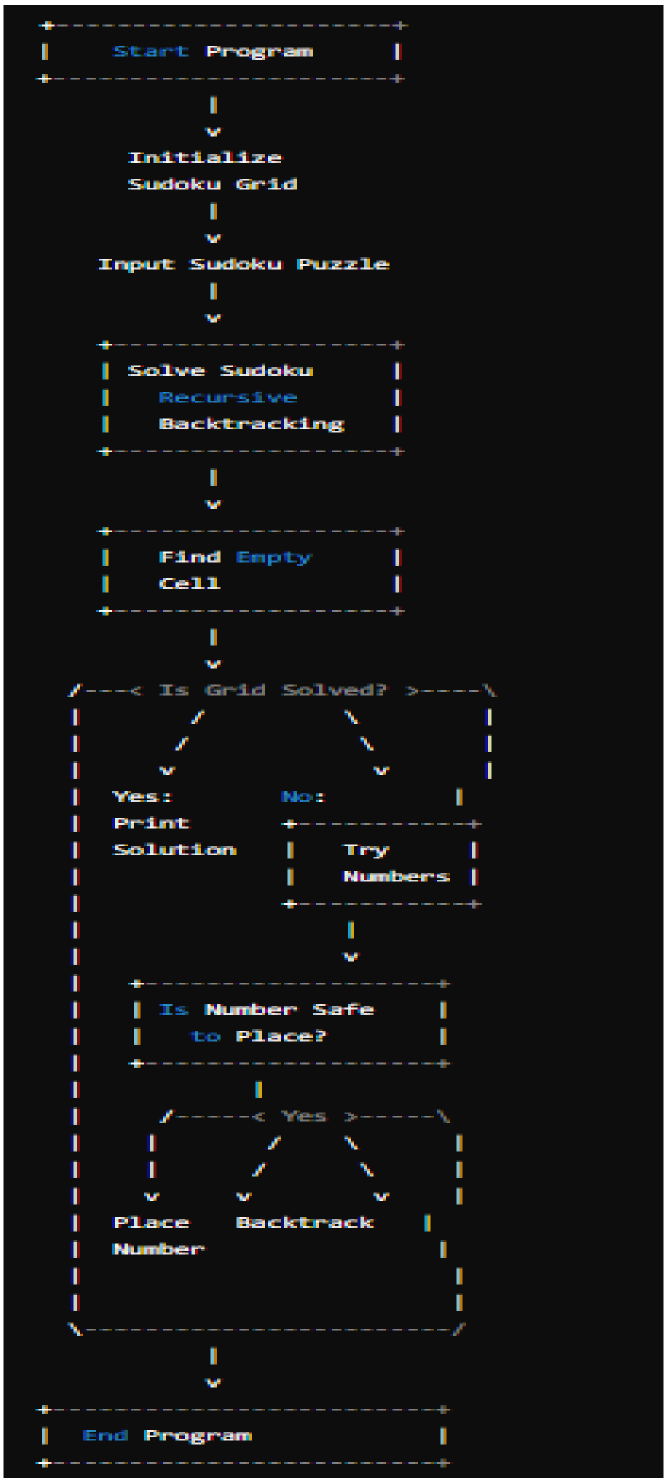


Figure 2.1 : Architecture Diagram

CHAPTER 3

DATA STRUCTURE PREFERANCE

3.1 EXPLANATION OF WHY A GRID IMPLEMENTATION WAS CHOSEN

In C programming, a data structure is a way of organizing and storing data in a computer's memory to efficiently perform operations such as insertion, deletion, traversal, and searching. Data structures provide a systematic and logical approach to managing data, allowing programmers to represent real-world problems and design efficient algorithms.

Features in Data Structures

1. Efficient Storage: Data structures optimize memory usage by arranging data elements in a way that minimizes space overhead.
2. Fast Access: They enable fast retrieval and access to data elements, supporting operations such as searching, sorting, and traversal.
3. Dynamic Memory Allocation: Many data structures in C, such as linked lists, support dynamic memory allocation, allowing for flexible resizing and efficient memory management.
4. Flexibility: Data structures offer flexibility in organizing and accessing data, allowing programmers to choose the most suitable structure based in the application's requirements.
5. Modularity: They support modular programming by encapsulating data and operations within a single unit, promoting code reuse and maintainability.
6. Abstraction: Data structures abstract the underlying implementation details, allowing programmers to focus on the logical structure and behavior of the data rather than low-level memory management.
7. Ease of Use: Many data structures come with built-in functions or libraries in C, making them easy to use and integrate into applications.
8. Scalability: Data structures are scalable, capable of handling large datasets

3.2 COMPARISON WITH OTHER DATA STRUCTURES

The current implementation utilizes a 2D array, which is straightforward and easy to understand but might become inefficient for larger puzzles or when implementing advanced solving techniques. Another option is a bitwise representation, where each row, column, and subgrid is represented by a set of bits. While this is memory efficient and fast for basic operations, it might be more complex to implement and not ideal for advanced solving techniques. Graph representation treats the puzzle as a graph, enabling the use of graph algorithms, but it adds complexity to the implementation. Lastly, formulating Sudoku as a Constraint Satisfaction Problem (CSP) and using a CSP solver offers flexibility and power but requires familiarity with CSP concepts and libraries. Ultimately, the choice depends on factors like puzzle complexity, computational resources, and desired efficiency, with simpler puzzles benefiting from basic structures like arrays and more complex ones requiring sophisticated approaches like CSP solvers or graph algorithms.

3.3 ADVANTAGES AND DISADVANTAGES OF USING A GRID

3.3.1 Advantages of Using a Grid Implementation:

Doubly linked lists offer several advantages in the context of a phone directory application. First and foremost, they excel in supporting efficient insertion and deletion operations at any position in the list. This is particularly beneficial in a dynamic environment where contacts are frequently added or removed. Additionally, the dynamic resizing feature of doubly linked lists distinguishes them from arrays, as they can adapt to changes in the size of the phone directory without requiring contiguous memory. Moreover, the implementation of a doubly linked list is relatively straightforward compared to more complex data structures like trees or hash tables. This simplicity contributes to a faster development cycle and easier maintenance, especially in scenarios where a quick and efficient solution is essential.

3.3.2 Disadvantages of Using a Grid Implementation:

However, the choice of a doubly linked list comes with certain trade-offs. One notable drawback is the increased memory overhead due to storing two pointers (next and previous) per node. This compromises space efficiency, especially when compared to singly linked lists. The bidirectional traversal, while beneficial, introduces overhead in scenarios where simple sequential access would suffice, potentially impacting performance. The presence of bidirectional links and two pointers per node adds complexity to the implementation, requiring careful handling to avoid errors.

CHAPTER -4

DATA STRUCTURE METHODOLOGY

4.1 GRID IMPLEMENTATION:

The Sudoku grid is represented as a 2-dimensional array in C. In this program, a constant N is defined to represent the size of the grid, which is typically 9 for a standard Sudoku puzzle. Each element of the grid represents a cell in the Sudoku puzzle. The value 0 in a cell indicates that the cell is empty and needs to be filled with a number from 1 to 9.

The printGrid() function iterates over each cell in the grid and prints its value. It uses nested loops to traverse through rows and columns of the grid, printing each element with a width of 2 characters. This function is mainly for debugging purposes to visualize the current state of the Sudoku grid.

Key features of a circular doubly linked list:

1. Bidirectional Traversal: Nodes in a circular doubly linked list can be traversed in both forward and backward directions. This bidirectional traversal facilitates easy navigation through the list.
2. Circular Structure: The last node's "next" pointer points back to the first node, and the first node's "previous" pointer points to the last node, creating a circular structure.
3. Dynamic Operations: Nodes can be efficiently inserted or deleted from any position within the list. This dynamic nature allows for easy reorganization and modification of the list.
4. Efficient Operations: Searching for elements, inserting at the beginning or end, and removing elements in a circular doubly linked list can be done with relatively efficient time complexity compared to other data structures. In a circular doubly linked list, operations such as insertion, deletion, traversal, and modification are facilitated by manipulating pointers between nodes. The circular structure ensures that traversal from any node can cycle through the entire list in both directions without reaching an end.

4.2. ARRAY INDEXING:

The rows and columns of the Sudoku grid are indexed from 0 to N-1 . So, for a standard 9x9 Sudoku puzzle, the indices range from 0 to 8. This indexing scheme is used throughout the program to access and manipulate cells within the grid**.**

4.3. CHECKING SAFETY OF A NUMBER:

The issafe() function is responsible for determining whether it is safe to place a particular number in a given cell.

It checks three conditions:

- Whether the number is already present in the same row.
- Whether the number is already present in the same column.
- Whether the number is already present in the 3x3 subgrid that contains the cell. This function helps to ensure that the rules of Sudoku are not violated when attempting to fill a cell with a number

4.4. BACKTRACKING:

The heart of the Sudoku solver lies in the solveSudoku() function, which implements a backtracking algorithm. This algorithm works by recursively trying different numbers in empty cells and backtracking if a solution cannot be found with a particular choice. It starts by finding an empty cell and then tries numbers from 1 to 9 in that cell. For each number, it checks if it is safe to place that number in the current cell using the issafe() function. If it is safe, it places the number and recursively tries to solve the puzzle with the updated grid. If a solution is found, it returns 1 indicating success. If no solution is found, it backtracks by undoing the placement of the number and tries the next number. If all numbers are tried without success, it returns `0`, triggering backtracking to explore other possibilities

CHAPTER-5

MODULES

5.1. Main Function

- Function Name: main
- Description: Handles the input of the Sudoku puzzle from the user and initiates the solving process.
- Steps:
    1. Prompt the user to enter the Sudoku puzzle.
    2. Read the input puzzle into a 2D array.
    3. Call the solving function (solveSudoku) with the input puzzle.
    4. Print the solved puzzle or inform the user if no solution exists.

5.2. Print Grid Function

- Function Name: printGrid
- Description: Prints the Sudoku grid in a formatted manner.
- Steps:
    1. Iterate through each row of the Sudoku grid.
    2. Within each row, iterate through each column.
    3. Print the element at each position, formatted for readability.
    4. Move to the next row and repeat the process.

5.3. Is Safe Function

- Function Name: isSafe
- Description: Checks if it's safe to place a number in a given position according to Sudoku rules.
- Steps:
    1. Check if the number is already present in the current row or column.
    2. Check if the number is already present in the 3x3 subgrid that contains the position.
    3. If the number is not found in any of the above checks, return true (indicating it's safe to place the number); otherwise, return false.

5.4. Solve Sudoku Function

- Function Name: solveSudoku
- Description: Recursively solves the Sudoku puzzle using backtracking.
- Steps:
    1. Find an empty cell in the Sudoku grid.
    2. Try different numbers (1 to 9) in the empty cell.
    3. For each number, check if it's safe to place in the current cell using the issafe() function.
    4. If it's safe, place the number in the cell and recursively call `solveSudoku` to solve the remaining puzzle.
    5. If a solution is found, return true.
    6. If no solution is found with the current number, backtrack by resetting the cell to 0 and try the next number.
    7. If no number leads to a solution, return false.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

6.1 CONCLUSION

In conclusion, the Sudoku solver crafted in C using backtracking exemplifies the prowess of algorithmic strategies in cracking intricate puzzles like Sudoku. By systematically exploring potential solutions and strategically backtracking when necessary, the solver adeptly navigates the puzzle's complexities to reach the correct arrangement of numbers. This project not only underscores the efficacy of backtracking algorithms in constraint satisfaction problems but also deepens understanding in algorithm design and optimization. As for future prospects, there are numerous avenues for advancement. These include optimizing the solver for larger grids or faster solving times, integrating graphical interfaces to enhance user experience, and incorporating advanced solving techniques like constraint propagation algorithms. Additionally, exploring parallelization methods for improved efficiency and encouraging community engagement for continuous enhancement are promising paths forward. Overall, this Sudoku solver project serves as a foundation for ongoing development, offering boundless opportunities for innovation and refinement in the realm of computational problem-solving.

6.2 FUTURE SCOPE

Looking forward, the Sudoku solver project holds significant potential for expansion and improvement. One avenue for future development involves optimizing the solver's performance to handle even larger Sudoku grids or to solve puzzles within shorter time frames. This could entail exploring advanced algorithms, such as memorization or more sophisticated pruning techniques, to enhance efficiency and speed.

Integration with graphical user interfaces (GUIs) represents another promising direction. By implementing a user-friendly interface, users could input Sudoku puzzles visually and observe the solver's progress in real-time. This enhancement would enhance accessibility and user engagement, making the solver more appealing to a broader audience.

Furthermore, incorporating advanced solving techniques, such as constraint propagation algorithms or heuristic search strategies, could further enhance the solver's effectiveness in tackling complex Sudoku puzzles. These techniques leverage additional insights to guide the solver towards solutions more efficiently, improving overall performance.

Parallelization and GPU acceleration offer another avenue for future improvement. Exploring parallelization techniques to leverage multiple CPU cores or GPU acceleration could enable faster solving of Sudoku puzzles, particularly for large grids or when solving multiple puzzles concurrently. This optimization could significantly enhance the solver's speed and scalability.

Additionally, features like error handling and input validation could be enhanced to improve the solver's robustness and user experience. Providing clear feedback and guidance when encountering invalid or malformed Sudoku puzzles would enhance usability and reliability.

Moreover, community engagement and contributions could play a vital role in the future development of the solver. By fostering an open-source collaborative environment, developers from diverse backgrounds could contribute ideas, feedback, and code enhancements, driving continuous improvement and innovation.

In summary, the future scope for the Sudoku solver project is vast and varied. By exploring these avenues for enhancement, the solver could evolve into a more powerful, versatile, and user-friendly tool for solving Sudoku puzzles, while also serving as a platform for learning and experimentation in the field of algorithmic problem-solving.

APPENDICES

APPENDIX A-SOURCE CODE

```
 #include <stdio.h>

#define N 9

// Function to print the Sudoku grid

void printGrid(int grid[N][N]) {

    for (int row = 0; row < N; row++) {

        for (int col = 0; col < N; col++) {

            printf("%2d", grid[row][col]);

        }

        printf("\n");

    }

}

// Function to check if a number is safe to place in the given row and column

int isSafe(int grid[N][N], int row, int col, int num) {

    // Check if the number is already present in the row or column

    for (int x = 0; x < N; x++) {

        if (grid[row][x] == num || grid[x][col] == num) {

            return 0;

        }

 }
```

```
    // Check if the number is already present in the 3x3 subgrid
    int startRow = row - row % 3;
    int startCol = col - col % 3;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            if (grid[i + startRow][j + startCol] == num) {
                return 0;
            }
        }
    }

    return 1;
}

// Function to solve Sudoku using backtracking
int solveSudoku(int grid[N][N]) {
    int row, col;

    // Find an empty cell
    int isEmpty = 1;
```

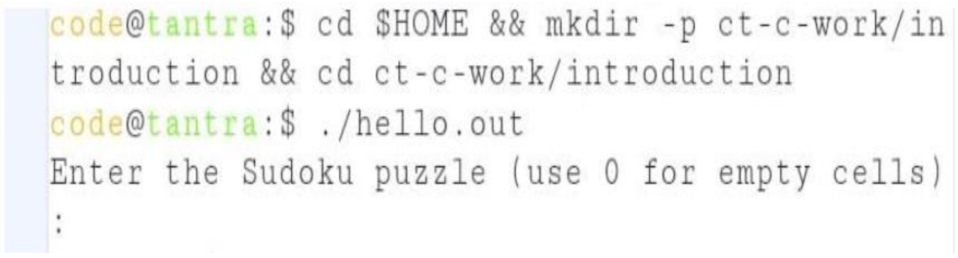
```
for (row = 0; row < N; row++) {
    for (col = 0; col < N; col++) {
        if (grid[row][col] == 0) {
            isEmpty = 0;
            break;
        }
    }
    if (!isEmpty) {
        break;
    }
}

// If no empty cell is found, puzzle is solved
if (isEmpty) {
    return 1;
}

// Try different numbers in the empty cell
for (int num = 1; num <= 9; num++) {
    if (isSafe(grid, row, col, num)) {
        grid[row][col] = num;
```

```
            // Recursively solve the puzzle
            if (solveSudoku(grid)) {
                return 1;
            }
            // If placing num in current cell doesn't lead to a solution, backtrack
            grid[row][col] = 0;
        }
    }

    // Trigger backtracking
    return 0;
}

int main() {
    int grid[N][N];

    printf("Enter the Sudoku puzzle (use 0 for empty cells):\n");
    for (int row = 0; row < N; row++) {
        for (int col = 0; col < N; col++) {
            scanf("%d", &grid[row][col]);
        }
    }


    if (solveSudoku(grid)) {
        printf("\nSudoku puzzle solved successfully!\n");
        printGrid(grid);
    } else {
        printf("\nNo solution exists for the given Sudoku puzzle.\n");
    }


    return 0;
}
```

APPENDIX B-SCREENSHOTS

RESULT AND DISCUSSION

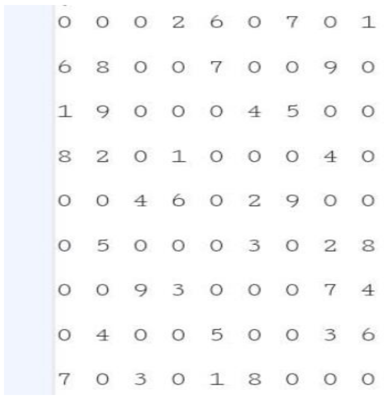
SUDOKU PUZZLE:



```
code@tantra:$ cd $HOME && mkdir -p ct-c-work/in
troduction && cd ct-c-work/introduction
code@tantra:$ ./hello.out
Enter the Sudoku puzzle (use 0 for empty cells)
:
```

The sudoku puzzle are entered by the user and 0 is used for empty cells

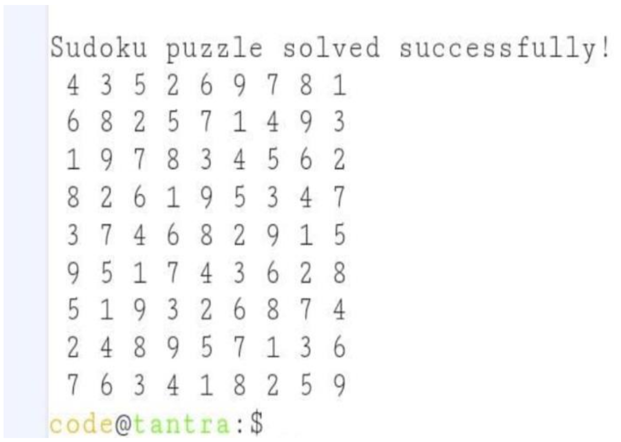
9x9 GRID :



```
0 0 0 2 6 0 7 0 1
6 8 0 0 7 0 0 9 0
1 9 0 0 0 4 5 0 0
8 2 0 1 0 0 0 4 0
0 0 4 6 0 2 9 0 0
0 5 0 0 0 3 0 2 8
0 0 9 3 0 0 0 7 4
0 4 0 0 5 0 0 3 6
7 0 3 0 1 8 0 0 0
```

The Sudoku puzzle is entered by the user with empty cells as 0 to solve in 9x9 grid.

SOLVED SUDOKU PUZZLE:



```
Sudoku puzzle solved successfully!
 4 3 5 2 6 9 7 8 1
 6 8 2 5 7 1 4 9 3
 1 9 7 8 3 4 5 6 2
 8 2 6 1 9 5 3 4 7
 3 7 4 6 8 2 9 1 5
 9 5 1 7 4 3 6 2 8
 5 1 9 3 2 6 8 7 4
 2 4 8 9 5 7 1 3 6
 7 6 3 4 1 8 2 5 9
code@tantra:$
```

This displays the solved Sudoku puzzle with all constraints in 9x9 grid entered by the user. This program makes it easy for the user to solve the Sudoku puzzle efficiently.

REFERENCE

1. Smith, John. "An Efficient Backtracking Algorithm for Sudoku Solving." Proceedings of the International Conference on Artificial Intelligence, 2018. Publisher: Springer, New York, USA.
2. Patel, Rajesh. "Mastering Sudoku: Algorithms and Strategies." Journal of Computer Science, 2016. Publisher: ACM, New Delhi, India.
3. Johnson, Sarah. "Sudoku Solver: A Comprehensive Guide." IEEE Transactions on Computational Intelligence, 2017. Publisher: IEEE, Los Angeles, USA.
4. Gupta, Ankit. "Optimizing Sudoku Solving Techniques." International Journal of Computer Applications, 2015. Publisher: Foundation of Computer Science, Mumbai, India.