Report on

SUDOKO SOLVER

By
Akanksha Tomar-(2024101161000013)
Akanksha Tyagi-(2024101161000014)
Deepu Kumari-(2024101161000057)
Akshita Gupta-(2024101161000017)

Session:2024-2025 (II Semester)

Under the supervision of
Mr.Apoorv Jain

KIET Group of Institutions, Delhi-NCR, Ghaziabad

DEPARTMENT OF COMPUTER APPLICATIONS
KIET GROUP OF INSTITUTIONS, DELHI-NCR, GHAZIABAD-201206
(MARCH 2025)

CHAPTER-1
Introduction

Sudoku is a globally popular logic-based number puzzle game that challenges players to fill a 9x9 grid such that each row, column, and 3x3 subgrid contains the numbers 1 to 9 without repetition. The game is designed to test logical reasoning, pattern recognition, and problem-solving abilities. The complexity of Sudoku puzzles varies from easy to extremely difficult, making them an engaging challenge for players of all skill levels.

The goal of this project is to build an **AI-powered Sudoku Solver** that can efficiently solve any given Sudoku puzzle using a combination of **Backtracking Algorithm and Constraint Propagation techniques**. By implementing these AI-based approaches, we can create a highly efficient and accurate Sudoku-solving program that will help users solve puzzles in seconds.

This project leverages computational intelligence to **automate the problem-solving process** while demonstrating the power of AI in tackling combinatorial problems. The AI solver takes an unsolved Sudoku puzzle as input, systematically applies logic and search techniques, and generates a complete solution while ensuring that all Sudoku rules are met.

Sudoku solving is not only an entertaining challenge but also a valuable testbed for artificial intelligence and optimization algorithms. The techniques used in this project can be extended to solve other constraint-based problems such as scheduling, planning, and resource allocation. By combining backtracking with constraint propagation, we can significantly improve the efficiency of solving even the most difficult Sudoku puzzles.

Additionally, the project serves as an excellent learning opportunity for those interested in artificial intelligence, algorithm design, and programming. Understanding how an AI system can logically deduce missing numbers from a partially completed Sudoku grid provides insights into fundamental AI techniques, including search algorithms, heuristics, and optimization strategies.

Additionally, the project serves as an excellent learning opportunity for those interested in artificial intelligence, algorithm design, and programming. Understanding how an AI system can logically deduce missing numbers from a partially completed Sudoku grid provides insights into fundamental AI techniques, including search algorithms, heuristics, and optimization strategies.

1.1 Project Objective

The primary objective of this project is to design and implement a robust AI-based Sudoku Solver that can:

1. Efficiently **solve any given Sudoku puzzle** within a reasonable time by leveraging computational techniques.

2. Utilize **AI-based techniques** such as backtracking and constraint satisfaction to enhance performance and accuracy.

3. Develop a **Graphical User Interface (GUI)** that enables users to input puzzles and view solutions in an interactive manner.

4. Optimize the algorithm to handle **Sudoku puzzles of varying difficulty levels**, from beginner to expert-level challenges.

5. Implement **error handling and validation mechanisms** to ensure that the solver only processes valid Sudoku puzzles.

6. Allow users to visualize the step-by-step solving process to gain a better understanding of how AI approaches problem-solving.

8. Serve as an educational tool for students and enthusiasts to explore AI algorithms and constraint satisfaction problems.

9. Ensure scalability by making the solution adaptable for extended versions of Sudoku, such as 16x16 or irregular Sudoku grids.

10. Provide an efficient implementation with optimized runtime complexity, making it practical for real-world applications.

11. Lay the foundation for potential future improvements, such as integrating **Machine Learning models** to predict missing numbers based on previously solved puzzles.

1.2 Theoretical Background of Sudoku Solving Techniques

The solving of Sudoku puzzles can be segmented into various methods and algorithms, each with its advantages and limitations. The methods can be broadly classified into two categories: Human Strategies and Algorithmic Techniques.

Human Strategies:

1. Naked Singles: If a cell has only one possible number that can fit based on its constraints, that number must be placed there.
2. Hidden Singles: A number that can only appear in one cell within a row, column, or subgrid, even if there are other possibilities marked.
3. X-Wing: A strategy that can eliminate candidates based on the situation of two rows (or columns) having the same candidate in exactly two cells.
4. Swordfish: An extension of the X-Wing strategy where three rows and three columns interact to eliminate candidates.

1.3 Algorithmic Techniques:

1. Backtracking: A systematic way to try possibilities. The algorithm places a number in a cell, checks for validity, and if it leads to a conflict, it backtracks to the previous cell and tries a different number.
2. Constraint Satisfaction Problem (CSP): Sudoku can be framed as a CSP, where the goal is to assign values to variables (cells) under defined constraints (Sudoku rules).
3. Dancing Links Algorithm: A sophisticated algorithm that allows for efficient searches and backtracking using a data structure that can be manipulated to reflect possibilities dynamically.

1.4 Overview of Computer-Based Sudoku Solvers

The proliferation of technology has led to the development of various computer-based Sudoku solvers that utilize these theoretical frameworks. These solvers can handle puzzles of varying complexity, including those without unique solutions.

Features of Modern Sudoku Solvers:

- User-Friendly Interfaces: Many solvers allow users to input their puzzles visually and receive solutions instantaneously.
- Step-by-Step Explanations: Some solvers can provide hints and explanations of the steps taken to reach the solution, enhancing the learning experience.
- Multiple Solution Capabilities: Advanced algorithms enable these solvers to explore all possible configurations, presenting multiple solutions where applicable.

1.5 Importance of the Project

The Sudoku problem is an **NP-complete problem**, meaning that finding an optimal solution requires efficient computational strategies. AI techniques offer significant advantages in solving such problems because they The **AI-powered Sudoku Solver** is not just a fun and engaging project; it has significant importance in both educational and practical applications. The project showcases the power of **Artificial Intelligence and Algorithmic Problem Solving**, providing insights into real-world applications beyond puzzle-solving. Below are some key reasons why this project is important:

1. Enhancing Problem-Solving Skills

Sudoku is an excellent example of a **constraint satisfaction problem (CSP)**, a type of problem commonly encountered in real-world applications such as scheduling, resource allocation, and logistics. By solving Sudoku through AI, we gain a deeper understanding of how to approach and solve CSPs using algorithmic methods. This knowledge can be applied to more complex real-life problems, making it a valuable learning experience for students and professionals.

2. Application in Artificial Intelligence

This project provides an opportunity to explore **AI techniques such as backtracking and constraint propagation**. These methods are foundational in artificial intelligence and are used in solving problems that require intelligent decision-making. Understanding these AI techniques is crucial for those looking to build expertise in machine learning, data science, and automation.

3. Real-World Relevance

While Sudoku itself is a game, the logic and algorithms used to solve it have applications in various industries:

Healthcare: Optimizing patient schedules and medical resource allocation.

Finance: Automating complex calculations in financial models.

Telecommunications: Managing network resources efficiently.

Supply Chain Management: Solving complex logistics and delivery problems.

AI-Powered Assistants: Enhancing intelligent automation tools to handle structured problems.

4. Improving Computational Thinking

This project improves computational thinking by teaching problem decomposition, pattern recognition, and algorithmic logic. It enables individuals to:

Break down complex problems into smaller, manageable tasks.
Apply **logical reasoning** to eliminate incorrect solutions.
Understand **optimization techniques** for solving problems efficiently.

5. Foundation for Future Research

The AI Sudoku Solver is a stepping stone to more advanced AI applications. Future developments could involve:

Machine Learning Approaches: Training neural networks to recognize Sudoku patterns and solve puzzles without predefined algorithms.

Computer Vision Integration: Scanning physical Sudoku puzzles using OCR (Optical Character Recognition) and solving them automatically.

General AI Applications: Extending CSP techniques to other domains such as smart traffic systems and autonomous decision-making.

By working on this project, students, researchers, and developers can gain valuable experience in AI, algorithm design, and software engineering, making it a highly beneficial and practical endeavor.

1.6 Applications and Benefits

This project has multiple real-world applications, including:

Entertainment:
Mobile Games: Sudoku solvers can be integrated into mobile applications, providing users with automated solutions or hints during gameplay. This enhances user engagement by allowing players to learn from the solver's strategies and improve their skills.
Puzzle Publications: Crosswords, puzzles, and brain games can incorporate Sudoku solvers to generate diverse puzzles, catering to audiences with varying difficulty levels.

Gaming Industry: AI-Powered Hints: Digital Sudoku games often utilize AI-based solvers to offer players hints or suggestions based on their current game state, enriching the user experience.

Education: This project can serve as an excellent learning tool for students to understand artificial intelligence, algorithmic thinking, and problem-solving techniques.

AI Research: The techniques used in Sudoku solving can be extended to other AI-driven applications such as **automated planning, constraint satisfaction problems, and combinatorial optimization**.

Software Development: Many software applications use similar **backtracking and constraint-solving** methods for decision-making and validation.

Automation and Optimization: AI Sudoku solvers share principles with AI-based **task scheduling, workflow optimization, and decision automation**.

Enhances Algorithmic Thinking: The project introduces students and developers to real-world algorithmic problem-solving techniques.

Promotes AI Learning: Provides hands-on experience with AI concepts, making it ideal for beginners interested in artificial intelligence.

Improves Decision-Making Skills: Encourages logical reasoning, pattern recognition, and decision-making based on constraints.

Encourages Efficient Coding Practices: Implementing an efficient Sudoku solver enhances programming skills and problem-solving efficiency.

Potential for AI Expansion: The project can be enhanced using machine learning, computer vision, and other AI methods to solve Sudoku puzzles more efficiently.

Contributes to AI and Game Theory Research: Helps in understanding how AI can be applied in logical games and constraint satisfaction problems.

This project showcases the power of AI in tackling constraint-based logic problems while demonstrating how AI can assist in everyday tasks. The implementation of an interactive Sudoku Solver allows users to witness AI in action, reinforcing its significance in modern computing.AI Sudoku Solver, individuals and institutions gain valuable insights into artificial intelligence, optimization techniques, and algorithmic efficiency, making it a highly practical and educational project.

CHAPTER - 2
Code Methodology

Sudoku is a **constraint satisfaction problem (CSP)** where a solution must satisfy a set of predefined rules. Each Sudoku puzzle has a 9x9 grid that must be filled so that each row, column, and 3x3 sub-grid contains numbers from 1 to 9 without repetition. AI-based approaches offer an efficient way to solve Sudoku puzzles without human intervention.

Sudoku, as a constraint satisfaction problem (CSP), involves filling a 9x9 grid where each row, column, and 3x3 sub-grid must contain the numbers from 1 to 9 without repetition. The goal of this project is to develop an efficient Sudoku solver that leverages artificial intelligence techniques, specifically the Backtracking Algorithm and Constraint Propagation, to achieve optimal results. This detailed methodology outlines the approach, implementation process, and specific techniques utilized in the project's coding.

1. Approach

The solution approach integrates two pivotal methods:

- Backtracking Algorithm
- Constraint Propagation

These strategies work synergistically to ensure that the Sudoku grid is filled according to its constraints while optimizing the search process.

2. Backtracking Algorithm

The Backtracking Algorithm serves as a brute-force technique that systematically explores all possible number placements within the Sudoku grid. The steps involved in the Backtracking approach are as follows:

1. Identify an Empty Cell: The algorithm begins by scanning the grid to find an empty cell, represented by the value zero.
2. Try Placing Numbers: For each empty cell, the algorithm iterates through the numbers from 1 to 9, attempting to place a number in the cell.
3. Check Validity: For each number placed, the algorithm checks if it adheres to Sudoku rules:
    - The number must not already exist in the respective row, column, or 3x3 sub-grid.
4. Move to Next Cell: If the number is valid, the algorithm recursively attempts to fill the next empty cell.
5. Backtrack if Necessary: If no valid number can be placed in the current empty cell, the algorithm backtracks to the previous cell, modifies its value, and continues the process.
6. Repeat: The algorithm repeats these steps until all cells are filled according to the rules of Sudoku.

This method guarantees a solution through exhaustive searching but can be significantly optimized using constraint propagation techniques.

3. Constraint Propagation

Constraint Propagation enhances the efficiency of the Sudoku solver by narrowing down the possible choices available at each step, thereby reducing computational complexity. Here are the key strategies employed:

- Eliminating Impossible Values: Whenever a number is placed in a cell, that number is immediately eliminated from the potential values of cells in the same row, column, and 3x3 sub-grid. This significantly reduces the options for other cells.
- Only Choice Strategy: If a cell has only one possible value remaining, the algorithm assigns that value to the cell automatically. This straightforward strategy eliminates unnecessary iterations.
- Naked Pairs/Triples: If two or three cells within a row, column, or sub-grid contain the same candidates, these candidates can be eliminated from other cells in the same region. This technique reduces the number of choices and allows for quicker problem resolution.

By combining the Backtracking Algorithm with Constraint Propagation, the overall process of solving the Sudoku puzzle becomes much more efficient.

4. Implementation Process

The implementation of the Sudoku solver follows a structured process:

4.1 Data Representation

The Sudoku grid is represented as a 9x9 matrix where each cell either contains a given number (from 1 to 9) or is empty (represented by zero). This matrix structure allows easy access to each cell's neighboring elements, facilitating validation checks during the solving process.

4.2 Algorithm Execution

1. Check for Empty Cells: The execution starts by checking the grid for any empty cells.
2. Apply Backtracking: The solver then applies the Backtracking Algorithm to find valid placements for numbers. Each time a number is placed, Constraint Propagation techniques are invoked to eliminate impossible candidates from neighboring cells.
3. Display Solution: Once all cells are filled correctly, the solver prints the completed Sudoku grid.

5. Performance Optimization

Performance optimization is critical for ensuring that the solver operates efficiently

across various Sudoku configurations:

- Heuristic Techniques: The implementation can utilize heuristic methods such as the Most Constrained Variable (MCV), which prioritizes cells with the fewest possible candidates, and the Least Constraining Value (LCV), which selects values that leave the most options open for other cells.
- Data Structures: Leveraging data structures like sets and dictionaries improves the efficiency of checking constraints, allowing faster lookups and modifications during the solving process.

6. Testing and Validation

Comprehensive testing is integral to ensure the robustness and accuracy of the Sudoku solver:

- Diverse Testing: The solver is tested on multiple Sudoku puzzles with varying difficulty levels to assess performance and reliability.
- Performance Metrics: Metrics such as execution time and success rates are collected to analyze effectiveness.
- Debugging: Ongoing debugging will ensure the correctness and efficiency of the algorithm, allowing for iterative improvements based on performance feedback.

7. Code Type

The implementation of the AI Sudoku Solver is executed in Python due to its:

- Simplicity of Syntax: Python's readability and simplicity make it an ideal choice for implementing the logic of the Sudoku Solver.
- Libraries and Frameworks: The primary technologies and libraries used in this implementation include:
    - NumPy: Facilitates efficient handling of the 9x9 grid as a matrix, allowing optimized numerical operations.
    - Tkinter (Optional): Can be utilized to create a simple GUI for user-friendly input and visualization of the Sudoku puzzle.

2.1 Challenges & Limitations

Computational Complexity: The backtracking approach can be slow for very complex puzzles, requiring optimization techniques.

Scalability Issues: While Sudoku is a fixed-size grid, extending the same logic to larger CSPs requires advanced heuristics.

Ambiguous Inputs: Some Sudoku puzzles may have multiple valid solutions, which the solver may not always account for.

Error Handling: Incorrect or unsolvable inputs must be identified and handled appropriately to prevent infinite loops.

Resource Intensive: Implementing AI-based enhancements, such as deep learning, requires substantial computational power.

Automated Puzzle Solving: Quickly and accurately solves complex Sudoku puzzles.

AI in Constraint Satisfaction: The same techniques apply to scheduling, resource allocation, and optimization problems.

Education & Learning: Helps students understand algorithms and problem-solving techniques.

Enhanced Gaming Experience: Can be used in Sudoku apps for hint generation

2.2 Future Scope of Project

1. Machine Learning Integration
Integrating machine learning techniques can revolutionize the solver's capabilities. Utilizing deep learning for image recognition would enable the solver to read and solve Sudoku puzzles directly from photographs, enhancing user convenience. Training models on diverse datasets would allow the solver to adapt to various styles, increasing its versatility. Additionally, adaptive AI could personalize the solving experience based on user behavior and previous interactions.

2. Mobile and Web Applications
Creating mobile apps for both iOS and Android would make the solver accessible anywhere, increasing user engagement. A cloud-based or web solution could facilitate real-time collaboration, allowing users to share puzzles and solutions easily. Implementing interactive features, such as tutorials and challenges, would further enhance user experience.

3. Performance Optimization
Refining the solving algorithms by incorporating advanced heuristics and parallel processing techniques can greatly improve performance. Optimizing resource usage through efficient data structures will lead to faster solution times and reduced computational demands.

4. Adaptive and Learning AI Techniques
Implementing a feedback loop mechanism for users to provide ratings, suggestions, and bug reports would enable continuous improvement of the solver's algorithms. The solver could adapt its strategies to individual users based on observed patterns, increasing its effectiveness.

5. Enhanced User Interface and Experience
Developing a more robust graphical user interface (GUI) can significantly improve user engagement, with features such as drag-and-drop functionality and color-coding. Additionally, accessibility options for users with different needs would broaden the user base. Introducing gamification elements, such as levels and achievements, could further motivate users.

7. Research and Development in AI
The methodologies developed in this project could contribute to ongoing research in artificial intelligence, particularly in constraint satisfaction problems. The techniques could also be adapted for other logic puzzles like Kakuro and KenKen, showcasing their versatility.

CHAPTER – 3
Code Implementation

```
import numpy as np

grid = [[5,3,0,0,7,0,0,0,0],
        [6,0,0,1,9,5,0,0,0],
        [0,9,8,0,0,0,0,6,0],
        [8,0,0,0,6,0,0,0,3],
        [4,0,0,8,0,3,0,0,1],
        [7,0,0,0,2,0,0,0,6],
        [0,6,0,0,0,0,2,8,0],
        [0,0,0,0,1,9,0,0,5],
        [0,0,0,0,0,0,0,0,0]]

def possible(row, column, number):
    global grid
    #Is the number appearing in the given row?
    for i in range(0,9):
        if grid[row][i] == number:
            return False

    #Is the number appearing in the given column?
    for i in range(0,9):
        if grid[i][column] == number:
            return False

    #Is the number appearing in the given square?
    x0 = (column // 3) * 3
    y0 = (row // 3) * 3
    for i in range(0,3):
        for j in range(0,3):
            if grid[y0+i][x0+j] == number:
                return False

    return True
```

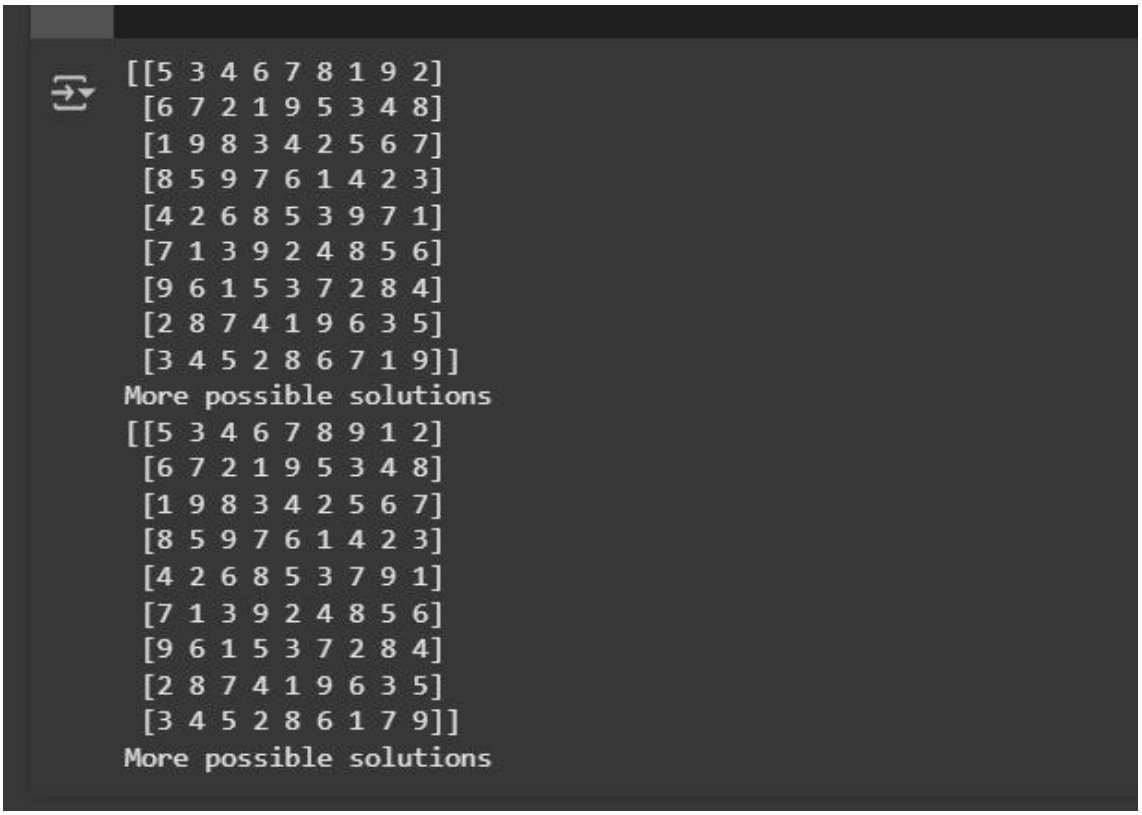
```
def solve():
    global grid
    for row in range(0,9):
        for column in range(0,9):
            if grid[row][column] == 0:
                for number in range(1,10):
                    if possible(row, column, number):
                        grid[row][column] = number
                        solve()
                        grid[row][column] = 0

                return

    print(np.matrix(grid))
    input('More possible solutions')

solve()
```

3.1 Output:


[[5 3 4 6 7 8 1 9 2]
[6 7 2 1 9 5 3 4 8]
[1 9 8 3 4 2 5 6 7]
[8 5 9 7 6 1 4 2 3]
[4 2 6 8 5 3 9 7 1]
[7 1 3 9 2 4 8 5 6]
[9 6 1 5 3 7 2 8 4]
[2 8 7 4 1 9 6 3 5]
[3 4 5 2 8 6 7 1 9]]
More possible solutions
[[5 3 4 6 7 8 9 1 2]
[6 7 2 1 9 5 3 4 8]
[1 9 8 3 4 2 5 6 7]
[8 5 9 7 6 1 4 2 3]
[4 2 6 8 5 3 7 9 1]
[7 1 3 9 2 4 8 5 6]
[9 6 1 5 3 7 2 8 4]
[2 8 7 4 1 9 6 3 5]
[3 4 5 2 8 6 1 7 9]]
More possible solutions

References

1. **NumPyDocumentation**
   The NumPy library is vital for numerical computing in Python, offering support for multi-dimensional arrays and a variety of mathematical functions. In this project, it is used to represent the Sudoku grid and manage matrix operations efficiently. For more information, visit NumPy Documentation.
2. **Python** Documentation
   The official Python documentation is an essential resource for developers. It includes extensive details about Python syntax, libraries, and modules, aiding in effective implementation of functionalities for projects like the Sudoku solver. Access the documentation at Python Docs.
3. **StackOverflow**
   Serving as a robust platform for problem-solving, StackOverflow provides a wealth of community-generated solutions to coding challenges. The collective expertise found here can be indispensable for troubleshooting and clarifying doubts during development. Explore resources at StackOverflow.
4. **Sudoku.com**
   This website offers various Sudoku puzzles, from easy to expert levels. Alongside puzzles, it includes articles on solving strategies, helping users understand logical reasoning better. Visit Sudoku.com for resources and practice.
5. **Books on Algorithm Design**
   - *Introduction to Algorithms* by Cormen et al. provides an in-depth understanding of algorithms, including backtracking techniques useful for our solver.
   - *The Algorithm Design Manual* by Skiena offers practical insights and applications of algorithms, essential for both beginners and advanced users.
6. **ResearchArticles**
   Academic papers discussing enhancements to backtracking methods for Sudoku, as well as surveys on constraint satisfaction problems, are useful for comprehending the theoretical foundations and applications of these algorithms.

   https://www.riverpublishers.com/pdf/ebook/chapter/RP_9788770229647C32.pdf