

Artificial Intelligence (CSE 2225) MINI PROJECT REPORT ON

**Sudoku Solver**

*SUBMITTED TO*

**Department of Computer Science & Engineering**

*by*

Name of the students

Reeva Nanda 220962310 Roll 48

Dev Vasudevan 220962378 Roll 60   
Laven Shivastava  
Aravind Shenoy

4th Semester

Name & Signature of Evaluator

(Jan 2024 - May 2024)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Table of Contents** | | | | |
|  | | | | Page No |
|  | | | | |
| **Chapter 1** | | | **INTRODUCTION** | 1 |
|  | **1.1** | Introduction | | 2 |
|  | **1.2** | Problem Statement | |  |
|  | **1.3** | Objective | |  |
|  | | | | |
| **Chapter 2** | | | LITERATURE REVIEW – Mention the topic from syllabus |  |
|  | **2.1** |  | |  |
|  | **2.2** |  | |  |
|  | | | | |
| **Chapter 3** | | | **METHODOLOGY – Discussion of algorithm and technique used** |  |
|  | **3.1** |  | |  |
|  | **3.2** |  | |  |
|  | | | | |
| **Chapter 4** | | | **RESULTS AND DISCUSSION** |  |
|  | **4.1** |  | |  |
|  | **4.2** |  | |  |
|  | | | | |
| **Chapter 5** | | | **CONCLUSIONS AND FUTURE ENHANCEMENTS** |  |
|  | **5.1** |  | |  |
|  | **5.2** |  | |  |
|  | | | | |
| **REFERENCES** | | | |  |
|  | | | |  |

**Rules for report making**

**Font – Times New Roman**

**TITLES Font Size – 14 pt BOLD**

**Subtitle – 12 pt BOLD**

Content – 12 pt.

**All figures and tables should be labelled.**

**Organization of Report:**

CONTENTS

1. Introduction

2. Literature survey, with brief descriptions of the contributions in each of the paper referred. / Course content from the syllabus

3. Methodology

4. Results and discussion

5. Conclusions and future enhancements, if any

6. Reference(s)

**Chapter 1: Introduction**

1.1 Introduction

Sudoku, a logic-based combinatorial number-placement puzzle, has gained immense popularity worldwide. It requires players to fill a 9x9 grid with digits so that each column, each row, and each of the nine 3x3 subgrids contain all of the digits from 1 to 9. The puzzle starts with some cells already filled, and the objective is to fill the remaining cells in a way that satisfies the Sudoku rules.

In this report, we present a Python-based Sudoku solver and generator. The project aims to provide a tool for Sudoku enthusiasts to solve puzzles and generate new ones of varying difficulty levels.

1.2 Problem Statement

The problem addressed in this project is twofold:

* Solving Sudoku puzzles: Given a partially filled Sudoku grid, the task is to fill in the remaining cells in a way that satisfies the Sudoku rules.
* Generating Sudoku puzzles: Creating new Sudoku puzzles of varying difficulty levels by partially emptying a complete Sudoku grid while ensuring the uniqueness of the solution.

1.3 Objective

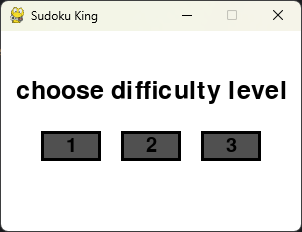
The primary objective of this project is to develop an efficient Sudoku solver and generator using Python. Specific goals include:

* Implementing a backtracking algorithm to solve Sudoku puzzles.
* Designing a method to generate Sudoku puzzles of different difficulty levels.
* Providing a user-friendly interface for interacting with the solver and generator.

The project aims to contribute to the field of recreational mathematics and provide a useful tool for Sudoku enthusiasts to enjoy and challenge themselves with puzzles of varying complexities.

[Placeholder for images of Sudoku puzzles or solved grids]

[Placeholder for code snippets demonstrating the solver and generator functionalities]



A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

**Chapter 2: Literature Review**

2.1) Overview of Depth First Search (DFS):

Depth First Search (DFS) is a fundamental algorithm used in graph traversal and searching. It explores as far as possible along each branch before backtracking. Here's an overview:

Algorithm:

Start from an initial node.  
Explore the deepest branch of the graph first.  
If a dead end is reached, backtrack to the most recent node with unexplored options.  
Repeat until all nodes are visited.

Key Characteristics:

DFS is implemented using recursion or a stack data structure.  
It's often used in solving puzzles, maze generation, and topological sorting.  
DFS can be applied to both directed and undirected graphs.

Advantages:

Simple to implement.  
Memory efficient compared to breadth-first search for large graphs.  
Suitable for solving problems with deep search trees.

Disadvantages:

May get stuck in infinite loops if not properly implemented.  
Not guaranteed to find the shortest path.  
Can be slow on graphs with many branches.

2.2) Overview of Backtracking:

Backtracking is a general algorithmic technique for finding solutions by incrementally building candidates and abandoning a candidate as soon as it determines that it cannot be completed to a valid solution. Here's an overview:

Algorithm:

Start with an empty solution and recursively try adding elements to it.  
If the current candidate is invalid, backtrack and try a different option  
Repeat until a valid solution is found or all possibilities are exhausted.

Key Characteristics:

Backtracking is often used to solve problems with a large search space, such as Sudoku puzzles, N-Queens, and maze solving.  
It systematically explores all possible solutions.  
Pruning techniques can be applied to reduce the search space and improve efficiency.

Advantages:

Guarantees to find a solution if one exists.  
Can handle problems with complex constraints.  
Memory efficient as it only explores one path at a time.

Disadvantages:

Can be slow if the search space is large.  
Requires careful implementation to avoid infinite loops.  
May not be the most efficient solution for some problems compared to other algorithms

**Chapter 3: Methodology**

3.1) Implementation of Depth First Search (DFS):

Initial State:

The initial state in DFS is the empty Sudoku grid, where some cells may already have pre-filled values.

Actions:

The actions in DFS involve placing numbers (1-9) into the empty cells of the Sudoku grid.  
The agent selects an empty cell and tries placing each possible number (1-9) into that cell.

Transition Model:

Each action involves placing a number into an empty cell.  
The transition model updates the Sudoku grid with the selected number at the chosen empty cell.  
If the placed number violates Sudoku rules, the transition model backtracks and explores a different option.

Goal:

The goal of DFS in the Sudoku solver is to fill all empty cells of the grid with valid numbers, satisfying the Sudoku rules.  
The DFS algorithm recursively explores the solution space until a valid solution is found.

Path:

The path in DFS represents the sequence of actions (placing numbers) taken to reach a solution.  
Each recursive call of the DFS function explores a branch of the solution space, backtracking when necessary.  
If a solution is found, the path consists of the sequence of numbers placed in each cell to reach the solution.

3.2) Implementation of Backtracking:

Initial State:

The initial state in Backtracking is the partially filled Sudoku grid, where some cells may already have pre-filled values.

Actions:

The actions in Backtracking involve placing numbers (1-9) into the empty cells of the Sudoku grid.  
Similar to DFS, the agent selects an empty cell and tries placing each possible number (1-9) into that cell.

Transition Model:

Each action involves placing a number into an empty cell.  
The transition model updates the Sudoku grid with the selected number at the chosen empty cell.  
If the placed number violates Sudoku rules, the transition model backtracks and explores a different option.

Goal:

The goal of Backtracking in the Sudoku solver is the same as DFS: to fill all empty cells of the grid with valid numbers, satisfying the Sudoku rules.  
Backtracking recursively explores the solution space until a valid solution is found.

Path:

The path in Backtracking represents the sequence of actions (placing numbers) taken to reach a solution.  
Each recursive call of the Backtracking function explores a branch of the solution space, backtracking when necessary.  
If a solution is found, the path consists of the sequence of numbers placed in each cell to reach the solution.