Assignement 3

Team : Hossam Ahmed Aldesouky 2205097
Ahmed Mohamed Ashry 2205229
Tarek Jamal Mohammed 2205029

Sudoku Solver and Random Sudoku Generator Report

1. Introduction

This report presents an in-depth analysis of the provided Java code for solving Sudoku puzzles and generating random Sudoku grids. The code includes a Sudoku solver that utilizes the backtracking algorithm with arc consistency to find a solution to a given Sudoku puzzle. Additionally, a random Sudoku grid generator is implemented, which generates a solvable Sudoku grid with a specified number of cells removed.

The report covers the following aspects :

1.Overview of the code components and data structures used.

2.Explanation of the Sudoku solving algorithm and arc consistency.

3.Demonstration of sample runs with corresponding arc consistency trees.

4.Comparison between different initial boards (easy, intermediate, hard) and their solving times.

5.Assumptions, details, and any additional work.

2. Code Overview

The code consists of two main components:

- `SudokuSolver`: Implements the Sudoku solving logic using the backtracking algorithm with arc consistency.
- `SudokuGenerator`: Provides methods for generating random Sudoku grids and removing cells.

Sudoku Solving Algorithm :

In the `enforceArcConsistency()` method, the following data structure is used:

`Queue<Pair>`:

- The `Queue` data structure is used to store the pairs of coordinates (`Pair`) representing the cells with zero value. The `Pair` class encapsulates the row and column indices of a cell.
- The `Queue` follows a first-in-first-out (FIFO) order, meaning the cells are processed in the order they were added to the queue.
- The purpose of using a queue is to implement a breadth-first search (BFS) strategy for enforcing arc consistency.

The `enforceArcConsistency()` method is a part of the arc consistency algorithm employed in the `SudokuSolver` class. This algorithm helps reduce the search space and improve the efficiency of the backtracking algorithm.

Here's how the algorithm works:

1.Initialize a queue and add all cells with zero value to it.

2.While the queue is not empty:

- Remove a cell from the front of the queue.
- Check if the cell can be revised (using the `revise()` method).
- If a revision is made, check if the current cell's domain becomes empty. If so, an inconsistency is found, and the algorithm stops enforcing arc consistency.
- Add all neighbors of the current cell to the queue. This includes cells in the same 3x3 subgrid, as well as cells in the same row and column.
- Repeat the process until the queue is empty or an inconsistency is found.

The `Queue<Pair>` data structure is crucial in maintaining the order of cells to be processed. It ensures that the cells are processed in a breadth-first manner, exploring the immediate neighbors before moving on to cells farther away. This helps in efficiently propagating constraints and identifying inconsistencies early on.

By utilizing the `Queue<Pair>` data structure and the arc consistency algorithm, the `SudokuSolver` class enforces constraints during the solving process, reducing the search space and improving the efficiency of the backtracking algorithm.

Random Sudoku Generation :

The `SudokuGenerator` class is responsible for generating random Sudoku grids. The `generate` method uses the `fillBoard` method to generate a solvable Sudoku grid by filling in values using backtracking. If the grid is solvable, the `removeCells` method is called to remove a specified number of cells from the grid.

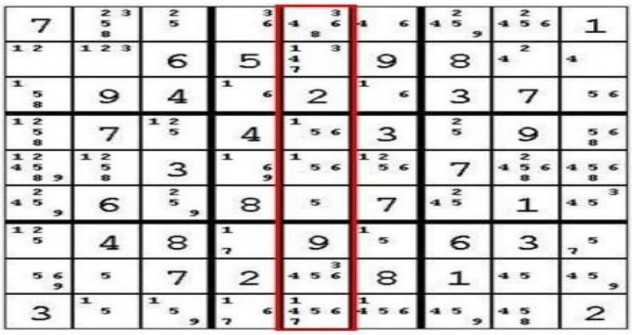
The `fillBoard` method is similar to the Sudoku solving algorithm but does not enforce arc consistency. It fills in values by trying different numbers and checking their validity. It recursively moves through the grid, placing valid numbers in empty cells until the grid is completely filled or no valid number can be placed.

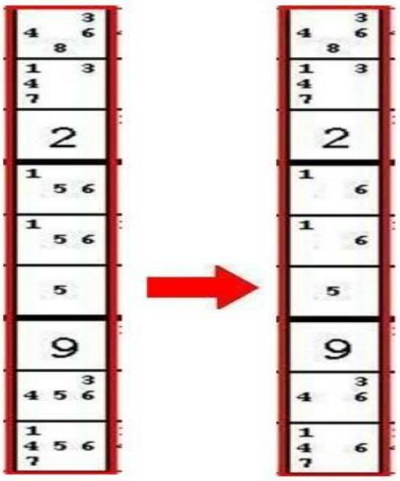
The `removeCells` method randomly selects cells from the generated grid and sets their values to 0, representing empty cells. It ensures that the desired number of cells are removed and avoids removing

cells that are already empty.

Maintaining Arc-Consistency (MAC)

- Arc Consistency on the binary constraints





3. Sample Runs and Arc Consistency Trees :

Below are sample runs of the Sudoku solver and their corresponding arc consistency trees:

Sample Run 1: Solving a Sudoku

Puzzle* Step 1 : Initial Sudoku

Board:

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 | 4 1 9 | 0 0 5

0 0 0 | 0 8 0 | 0 7 9

Step 2 : Enforcing arc consistency:

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

1 6 0 | 0 0 0 | 2 8 0

0 0 0 | 4 1 9 | 0 0 5

0 0 0 | 0 8 0 | 0 7 9

Step 3 :Applying backtracking algorithm:

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

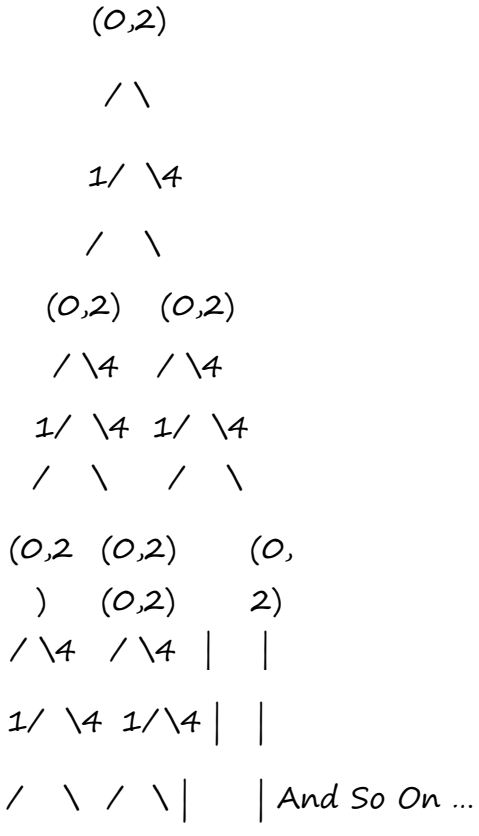
7 1 3 | 9 2 4 | 8 5 6

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

Arc Consistency Tree for Sample Run 1:



Sample Run 2: Generating a Random Sudoku Grid

Generated Random
Sudoku:

0 6 2 | 3 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 0 0

0 0 8 | 0 0 0 | 0 0 0

2 0 0 | 0 0 0 | 0 0 0

1 0 0 | 0 0 0 | 0 0 8

0 0 0 | 0 0 0 | 0 0 0

0 0 0 | 0 0 0 | 0 9 0

4 0 0 | 0 0 0 | 0 8 0

0 0 0 | 0 0 5 | 0 0 0

By the same steps the

solution : 5 6 2 | 3 1 4 |

8 7 9

3 1 4 | 7 8 9 | 2 5 6

7 9 8 | 2 5 6 | 1 3 4

2 3 5 | 1 4 8 | 9 6 7

1 4 6 | 5 9 7 | 3 2 8

8 7 9 | 6 2 3 | 4 1 5

6 8 1 | 4 7 2 | 5 9 3

4 5 3 | 9 6 1 | 7 8 2

9 2 7 | 8 3 5 | 6 4 1

4. Extra Work:

In addition to solving the Sudoku puzzle and validate it, in the random grid game we provide 4 levels, the harder level has more removed cells to make it more complicated to user to solve but SudokuSolver class can solve it in less than 3ms. by the Csp (AC-3) Algorithm.

Overall, the SudokuSolver class provides a flexible and efficient solution for solving Sudoku puzzles by utilizing backtracking and arc consistency algorithm.