**Report: Solving Sudoku Puzzle Using Backtracking and Constraint Propagation**

**Introduction**

Sudoku is a popular puzzle game where the objective is to fill a 9x9 grid with numbers such that each row, column, and 3x3 sub-grid contains all the digits from 1 to 9. The game is known for its difficulty in finding a unique solution, and often requires a combination of logic and trial-and-error to solve.

In this project, we have implemented two algorithms to solve Sudoku puzzles: backtracking and constraint propagation. The backtracking algorithm is a brute-force approach that tries every possible value for each empty cell until a valid solution is found. The constraint propagation algorithm, on the other hand, uses a set of rules to eliminate invalid values for each cell and gradually narrows down the possible solutions.

**Code Description**

The code is written in Python and contains two main functions: solve\_backtracking and solve\_propagation. The solve\_backtracking function uses a recursive backtracking approach to solve the puzzle, while the solve\_propagation function uses a constraint propagation approach.

The code begins by defining some constants such as the grid size, the number of cells to fill in for the --hint flag, and the set of possible values for each cell. It then defines a function is\_valid to check if the input grid is a valid Sudoku puzzle. This function checks that each row, column, and 3x3 sub-grid contains unique values.

The next function defined is find\_empty which finds the next empty cell in the grid. If no empty cells are found, the puzzle is solved.

The solve\_backtracking function is then defined which recursively fills each empty cell with a possible value and checks if the solution is valid. If the solution is not valid, it backtracks and tries the next possible value. If a valid solution is found, it is returned. If no valid solution is found, the function returns None.

Finally, the solve\_propagation function is defined which uses a set of rules to propagate the constraints and eliminate invalid values for each cell. This function first creates a set of sets to represent the possible values for each cell. It then initializes the possible values based on the input grid. The function then defines a propagate function which propagates the constraints until the puzzle is solved or an unsolvable cell is found. The function returns the solved puzzle or None if no solution is found.

**Conclusion**

In conclusion, we have implemented two algorithms to solve Sudoku puzzles: backtracking and constraint propagation. The backtracking algorithm is a brute-force approach that tries every possible value for each empty cell until a valid solution is found. The constraint propagation algorithm uses a set of rules to eliminate invalid values for each cell and gradually narrows down the possible solutions. Both algorithms have their strengths and weaknesses, and the choice of which algorithm to use depends on the specific problem at hand.