Sudoku is a classic puzzle game based in logic, where you have to place numbers in different combinations. The ultimate goal of the game is to complete a 9x9 grid so each row, column, and smaller 3x3 subgrids contain all of the digits from 1 to 9. This report will describe the approaches taken and the reasoning behind them in relation to the development of code that solves increasingly difficult Sudoku puzzles.

The starting basis for my algorithm was a simple backtracking approach. In essence, this is a trial and error method that attempts to place the Sudoku numbers, progressing when a correct number is found and moving back a step to a previous decision when it reaches a dead end (Russell and Norvig, 2021). This works well for the “very easy” and “easy” puzzles, however, as the puzzles grew in difficulty a more advanced and intricate strategy. The idea was to gradually solve puzzles while trialing more advanced techniques within the algorithm to test their efficacy.

The backtracking part of the algorithm places numbers in the Sudoku grid one at a time and moves left to right and top to bottom. In each empty cell, it tries digits from 1 to 9 and if the number doesn’t break Sudoku’s rules, it is placed in the empty cell. The algorithm then moves on to the next empty cell and starts again, if it finds a cell where no number is acceptable, it “backtracks” to the cell prior and attempts the next number in the order. This will continue until the puzzle is solved or until all possibilities have been tested. This is combined with another approach called the “Most Constrained Variable Heuristic” which means choosing the variable with the fewest options left. As typical Sudoku players will fill in numbers sequentially, however, using this approach will target cells with the least possible numbers to reduce the search space and find speedier solutions.

To solve the “medium” and “hard” puzzles I needed to employ some more advanced techniques which ranged from naked and hidden singles to naked and pointing pairs. In addition, I used a dictionary to track possible numbers. In the find_next_cell function, I identified the naked cells with only one possible number and filled them in right away. Next, I filled in the hidden singles that were only viable within a column, row, or 3x3 grid. Overall, these constraint propagation techniques streamlined the search by reducing the available space. (Russell and Norvig, 2021). I then employed an eliminate_naked_pairs function which finds cells that share two digits and eliminates them from the rest of the grid. My eliminate_pointing_pairs function focuses on pairs in a column or row within a grid, and then excludes them from other cells in their row or column. (Russell and Norvig, 2021). Next, I established a dictionary called “candidates” which keeps track of the numbers for each cell so the algorithm didn’t have to continuously recheck numbers. Sutton and Barto (2018) support this when they write about the need for data structures in order to maintain algorithmic efficiency.

Lastly, I considered and tested several advanced techniques focusing on elimination across columns, rows and grids, to further improve the efficacy of the algorithm. Ranging from X-Wing and Y-Wing to Swordfish and Jellyfish techniques. All of these could’ve provided ample opportunities in terms of constraint propagation and the reduction of search space (Russell and Norvig, 2021). It would be fascinating to explore and implement these methods, even through training models on Sudoku puzzles to predict answers through reinforcement learning as cited by Sutton and Barto (2018).

References:

- Russell, S. J., & Norvig, P. (2021). Artificial intelligence: A modern approach (4th ed.). Pearson.
- Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). The MIT Press; Bradford Books.