Sudoku Solver Report

Initial Approach: MRV and Backtracking

My initial Sudoku solver used **Most Constrained Variable (MRV) heuristics** paired with **constraint propagation and backtracking**, solving puzzles in about **3 seconds**. The MRV heuristic prioritized cells with the fewest possibilities, effectively reducing branching. However, this solver dynamically recalculated constraints at every step, which introduced performance bottlenecks as the puzzle complexity increased.

Transition to SudokuGameSolver

To address these limitations, I developed **SudokuGameSolver**, which achieved a **100x improvement** in performance, solving puzzles in just **0.03 seconds**. This leap was accomplished through several critical optimizations:

1. Constraint Mapping

I precomputed a detailed map linking every cell and value to their constraints (row, column, and subgrid). This eliminated repetitive checks by enabling instant conflict detection, which significantly reduced computation time.

2. Precomputed Constraint Sets

All valid possibilities for each cell were calculated at the start. This removed the need for dynamic recalculations during the solving process, allowing the solver to focus solely on decision-making and conflict resolution.

3. Efficient Backtracking

SudokuGameSolver stored and restored constraints during backtracking instead of recalculating them dynamically. This streamlined the rollback process, making it faster and more efficient.

4. Modular Design

The solver's logic was modularized into distinct components for constraint management, value placement, and conflict resolution. This made the code cleaner and easier to maintain while boosting execution efficiency.

Results and Reflections

With SudokuGameSolver, execution time dropped from **3 seconds to 0.03 seconds**, alongside better memory efficiency and streamlined constraint handling. These optimizations highlight the power of precomputation and efficient data structures in enhancing algorithmic performance.

In the future, I aim to explore parallel computation and AI-driven heuristics to tackle even more complex puzzles. This journey underscored the importance of thoughtful design, modularity, and precomputation in creating scalable and efficient solutions

References

Russell, S. and Norvig, P. (2020) Artificial intelligence: A modern approach. 4th edn. Pearson.

Simonis, H. (2005) ‘Sudoku as a constraint problem’, in Proceedings of CP 2005: International Conference on Principles and Practice of Constraint Programming. Springer, pp. 13–27.

Van Hentenryck, P. et al. (1998) Constraint processing. MIT Press.

Kaggle (n.d.) ‘Sudoku puzzles dataset’, Kaggle. Available at: https://www.kaggle.com/datasets/bryanpark/sudoku