Sudoku Solver

Code Report

Artificial Intelligence

CM12001

January 8, 2024

1 Difficulty of Approach

The Sudoku solver developed for this project presents a balanced approach, embodying both moderate complexity and methodical ambition. It incorporates two core techniques: constraint propagation and recursive depth-first search coupled with time-based monitoring. The sophistication of the algorithm is further enhanced by strategic optimizations. These include employing two Sudoku solving strategies, utilizing two search optimization heuristics, and leveraging efficient data structures for effective game representation and operation. Although each technique, when considered independently, is relatively straightforward, the overall complexity of the algorithm arises from numerous interactions between the mechanisms, many of which are of recursive nature. The intricacies of the described techniques and optimizations are elaborated in Sections 2 and 3.

2 Description of Algorithm

The game states are represented by the `grid` dictionary, with cell coordinates as keys, and strings of available digits as values [1]. The algorithm is based on two primary concepts: constraint propagation and recursive depth-first search (RDFS). The former is leveraged to iteratively and recursively eliminate available digits for each cell based on Sudoku rules. For instance, when a cell is determined to have only one valid digit, this digit is then recursively removed from the list of possible values for all cells in the cell's `peers`. This action can trigger further changes in the `peers` of these `peers`, effectively narrowing down the search space. However, to ensure algorithmic completeness, constraint propagation is complemented by RDFS, which selects the optimal cell and digit for elimination using heuristics. Each decision initiates a new search branch. If a branch leads to a dead end, the `ContradictionException` is raised, the branch is immediately discarded, and another path is explored. This process continues until a solution is found or all possibilities are exhausted, ensuring a complete and efficient solution.

3 Optimizations and Complexity

While the space complexity remains constant due to the fixed size of the grid and the data structures employed to represent it, the theoretical time complexity is calculated as $O(9^n)$, where n is the number of empty spaces, and 9 is the branching factor (the number of possible digits) [2]. However, in practice, this exponential complexity is significantly mitigated by the implementation of strategic heuristics and optimizations. The minimum remaining values heuristic selects the cell with the fewest available digits, thereby reducing the average branching factor. The least constraining values heuristic further refines the digit selection by choosing a digit that affects the least number of the cell's `peers`. Furthermore, the *hidden singles* strategy assigns digits to cells where only one valid option exists within a `unit`, and the *naked twins* strategy eliminates specific digits from peers within a `unit`, based on pairs of cells holding identical digit pairs. Lastly, a time constraint of 29.99 seconds is applied, after which `INVALID_SOLUTION` is returned. This approach is based on empirical observations suggesting that puzzles requiring more than this duration are typically either too complex to resolve within a reasonable time frame or lack a viable solution.

4 Reflections and Suggestions for Further Work

Although the algorithm demonstrates notable efficiency, it occasionally encounters puzzles that challenge the predefined time threshold (see Section 3), suggesting there is still room for improvement in terms of strategies, optimizations and implementation methods. Future enhancements might focus on integrating advanced heuristics or applying machine learning algorithms to refine the search strategy. Exploring parallel processing could offer a significant speedup by concurrently exploring multiple solution branches. Additionally, adjusting the time threshold dynamically based on puzzle complexity could result in more nuanced management of edge cases. Extensive testing across various puzzle difficulties could further enhance the algorithm's robustness and overcome its limitations.

References

[1] Norvig, P. *Solving Every Sudoku Puzzle* [Online]. Available from: `https://norvig.com/sudoku.html`. See the rationale for using strings in `grid` and data structures for `units` and `peers` under the sections *Sudoku Notation and Preliminary Notions* and *Analysis*.

[2] AfterAcademy, 2020. *Sudoku Solver* [Online]. Available from: `https://afteracademy.com/blog/sudoku-solver/`. See the *Complexity Analysis* section.